VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Využití OLED displeje v biomedicínské číslicové technice

Use of OLED Displays in Biomedical Digital Technique

2018 **Marta Ševčáková**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zadání bakalářské práce

Student: **Marta Ševčáková**

Studijní program: B2649 Elektrotechnika

Studijní obor: 3901R039 Biomedicínský technik

Téma: Využití OLED displeje v biomedicínské číslicové technice
Use of OLED Displays in Biomedical Digital Technique

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Rozbor technologie OLED displejů, vlastností a použití.
2. Seznámení se s technikou FPGA a návrhem programovatelných logických obvodů.
3. Návrh demonstrační úlohy s barevným maticovým OLED displejem do oblasti biomedicínské techniky.
4. Vytvoření demonstrační úlohy s využitím programovatelné architektury Zynq.
5. Experimentální ověření úlohy na vývojové desce ZYBO.
6. Zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

[1] ANDERSON, Paul. *Advanced Display Technologies.* JISC Technology & Standards Watch [online] [cit. 30.6.2017]. Dostupné z: https://pdfs.semanticscholar.org/1dda/2d638a42a6ecea239ec2dc72f196f4111fb4.pdf.
[2] CROCKETT, Louisse H., Ross A. ELLIOT, Martin A. ENDERWITZ a Robert W. STEWART. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc.* Glasgow: Strathclyde Academic Media, 2014. ISBN 978-0992978709. Dostupné také z: http://www.zynqbook.com/.
[3] XILINX, Inc. *Vivado Design Suite Tutorial* [online]. [cit. 30.6.2017]. Dostupné z: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug995-vivado-ip-subsystems-tutorial.pdf
[4] DIGILENT, Inc. *PmodOLEDrgb: : 96 x 64 RGB OLED Display with 16-bit color resolution* [online] Dostupné z: http://store.digilentinc.com/pmod-oledrgb-96-x-64-rgb-oled-display-with-16-bit-color-resolution/.
[5] KAPPENMAN, Tommy. Getting the PmodOLEDrgb to Work on Zybo. In: *Instructables* [online] [cit. 30.6.2017]. Dostupné z: http://www.instructables.com/id/Getting-the-PmodGPS-to-Work-on-Zybo/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Vladimír Kašík, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jiří Koziorek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě dne: *27. dubna 2018*

Řezáčová Marta

.....

podpis studenta

Poděkování

Především bych touto cestou ráda poděkovala vedoucímu panu *Ing. Vladimíru Kašíkovi, Ph.D.* za odborné vedení, velmi cenné rady a věnovaný čas, při vytváření této bakalářské práce.

Abstrakt

Práce je sestavena ze tří částí, které přechází vždy od teoretického základu ke konkrétně použité součástce v ukázkové úloze. První úsek se zabývá rozborem architektury, funkcionality, typů, vlastností, využití a zhodnocení elektroluminiscenčních elektronických součástek (diod), jež obsahují organickou vrstvu mezi elektrodami (OLED). Druhý tematický celek se zaměřuje na programovatelné logické obvody. Opět od stručného nastínění problematiky programovatelných hradlových polí a jejich návrhu, přechází přes parametry použité FPGA desky ZYBO k její SoC architektuře Zynq 7000. Klíčovou část práce tvoří ukázková úloha, na které je demonstrován průběh prvotního návrhu, přes vytváření úlohy s pomocí souboru návrhových nástrojů firmy Xilinx, sloužících k vytvoření jak softwarové, tak hardwarové části programující procesor desky programovatelných hradlových polí, až k samotnému ověření funkceschopnosti vytvořeného programu na poskytnutých komponentách ZYBO Zynq 7000 a barevném organickém LED displeji PmodOLEDrgb. Dosažené výsledky reprezentují detailně popsané funkce a čtyři animace prostředí aplikací zdravotnických přístrojů. Celkově tato bakalářská práce vytváří přehled aspektů návrhu a vytváření animací na OLED displeji řízeném FPGA.

Klíčová slova

OLED; Organické světlo emitující diody; FPGA; Programovatelná hradlová pole; PLD; Programovatelné logické obvody

Abstract

The thesis is composed of three main parts, which begins with a base theoretical knowledge of the relevant components used in the demonstrational task. The first part deals with the analysis of architecture, functionality, types, properties, utilization and evaluation of electroluminescent electronic components; they contain the organic layer between the two electrodes. The second thematic unit focuses on programmable logic circuits. From a brief drawing of the problem of programmable gate arrays and their design, it passes through the parameters of the ZYBO FPGA boards to its Zynq 7000 SoC architecture. Lastly, a sample task is performed that demonstrates the course of the initial design by utilizing Xilinx design tools. These tools create both the hardware and software that programs the processor of the programmable gate arrays provided by the ZYBO Zynq 7000 components and the PmodOLEDrgb colour LED organic display. The results obtained represent four major visualizations of medical device applications. Overall this bachelor thesis creates an overview of aspects of designing and creating animations on OLED display powered and driven by FPGA.

Key words

OLED; Organic light-emitting diode; FPGA; Field Programmable Gate Array; PLD; Programmable logic device

Obsah

Seznam použitých zkratek

Zkratka	Anglický význam	Český význam
ACP	*Accelerator Coherency Port*	-
AMOLED	*Active-matrix OLED*	OLED s aktivní maticí
APU	*Application processing unit*	Čipová jednotka
ASIC	*Application Specific Integrated Circuits*	Zákaznický integrovaný obvod
BMP	*Bitmap*	Bitová mapa (rastrový obrázek)
CLB	*Configurable Logic Block*	Konfigurovatelný logický blok (buňka)
CPU	*Central processing unit (processor)*	Centrální procesorová jednotka
CRT	*Cathode Ray Tube*	Displej s katodovou trubicí
FIFO	*First In First Out*	První dovnitř, první ven
FOLED	*Flexible OLED*	Ohebný OLED
FPGA	*Field Programmable Gate Array*	Programovatelná hradlová pole
GDDRAM	*Graphic Display Random-Access Memory*	Paměť RAM grafického displeje
GPIO	*General Purpose Input/Output*	Obecné vstupně výstupní bloky
HDL	*Hardware Description Language*	Jazyk pro popis hardwaru
IO	*Integrated circuit*	Integrovaný obvod
IOB	*Input/Output Block*	Vstupně/výstupní bloky
LCD	*Liquid Crystal Display*	Displej z tekutých krystalů
LED	*Light Emitting Diode*	Světlo emitující dioda
LEP	*Light Emitting Polymers*	Světlo vyzařující polymery
LUT	*Look-up table*	Náhledová tabulka
MOSI	*Master Out, Slave In*	Datový výstup SPI rozhraní
MISO	*Master In, Slave Out*	Datový vstup SPI rozhraní
OCM	*On Chip Memory*	Paměť na čipu
OLED	*Organic Light Emitting Diode (Device)*	Organická světlo emitující dioda
PHOLED	*Phosphorescent OLED*	Fosforeskující OLED
PL	*Programmable logic*	Programovatelná logika
PLD	*Programmable logic device*	Programovatelné logické obvody
PMOLED	*Passive-matrix OLED*	OLED s pasivní maticí
PS	*Processing System*	Procesorový systém
PWM	*Pulse Width Modulation*	Pulzně šířková modulace

RAM	*Random Access Memory*	Paměť s náhodným přístupem
RGB	*Red Green Blue (colour spectrum)*	Červená-zelená-modrá (barevné spektrum)
RTL	*Register Transfer Level*	Meziregistrové přenosy
SCLK	*Signal Clock*	Hodinový signál
SCU	*Snoop Control Unit*	-
SDK	*Software development kit*	Soubor nástrojů pro vývoj softwaru
SMOLED	*Small Molecule LED*	LED s malými molekulami organické látky
SOLED	*Stacked OLED*	Vrstvený OLED
SPI	*Serial Peripheral Interface*	Sériové periferní rozhraní
STA	*Static Timing Analysis*	Statická časová analýza
SS	*Slave Select*	Výběr slave zařízení
TFT	*Thin Film Transistor*	Tenkovrstvé tranzistory
TOLED	*Transparent OLED*	Průhledný OLED
VHDL	*VHSIC Hardware Description Language*	Programovací jazyk hardwaru
VHSIC	*Very High Speed Integrated Circuit*	Vysokorychlostní integrovaný obvod
WOLED	*White OLED*	Bílá OLED
ZYBO	*Zynq Board*	FPGA deska s architekturou Zynq

Seznam obrázků

Seznam tabulek

Úvod

Displeje tvoří nenahraditelnou součást většiny lékařských přístrojů a zařízení. A vzhledem k tomu, že v současnosti využití organických materiálů v elektronice patří k nejkvalitnějším a nejperspektivnějším zobrazovacím technologiím na trhu, tak tato kombinace nabízí zkvalitnění a usnadnění lékařské péče. Aktuálně jsou programovatelná hradlová pole uplatňována v široké škále různých aplikací. Pořizovací cena FPGA obvodů klesá a nachází široké uplatnění díky možnosti zcela individuálního nastavení vnitřní struktury pro jakoukoli aplikaci, krom toho další výhodou je možnost zpracování dat paralelně tzn. v jeden časový okamžik, tím se podstatně zkrátí doba potřebná k vykonání požadovaných operací, což se dá využít k zrychlení zobrazování snímků získaných prostřednictvím jakýchkoli lékařských zařízení.

Práce obsahuje dva tematické celky, které přechází vždy od stručného teoretického základu k parametrům konkrétně použité komponenty v ukázkové úloze. První úsek se zabývá rozborem architektury, funkcionality, typů, vlastností, využití a zhodnocení světlo emitujících elektronických součástek, jež obsahují organickou vrstvu mezi elektrodami. Druhý tematický celek se zaměřuje na programovatelné logické obvody. Od stručného popisu struktury a funkce programovatelných hradlových polí a jejich návrhu, přechází přes parametry použité FPGA desky k její SoC architektuře. Demonstrační úloha tvoří třetí důležitou část práce. Teoretický úvod demonstrační úlohy se skládá z náhledu do průběhu návrhu a programování. Vytváření programu probíhá ve dvou hlavních krocích nejprve programování hardwarové části, tedy návrh do programovatelných hradlových polí FPGA, a poté softwarové části programu, která byla psána v programovacím jazyce C.

Praktická část se věnuje konkrétnímu využití kombinace těchto technologií, demonstrované na názorných příkladech, realizovaných na desce programovatelných hradlových polí typu ZYBO s architekturou Zynq 7000 komunikujícím přes konektor Pmod a komunikační rozhraní SPI s barevným OLED displejem typu PmodOLEDrgb. Část obsahu práce je věnována rozboru principu řízení jednotlivých komponent. Nabízí seznámení s parametry využitého barevného displeje PmodOLEDrgb a nástin jeho řízení řadičem typu SSD1331. Dále pak náhled do struktury architektury SoC Zynq a potažmo celé vývojové desky ZYBO, která tento čip obsahuje. Ovšem většina kapitoly se zabývá návrhem a samotným vytvořením demonstrační úlohy, které bylo z velké části realizováno ve vývojovém prostředí Vivado a SDK, jež patří do souboru návrhových nástrojů firmy Xilinx.

Cílem bakalářské práce je nastínění techniky barevných organických LED displejů, programovatelných hradlových polí a seznámení se samotným návrhem programovatelných logických obvodů. Hlavní část tvoří ukázková úloha, na které jsou demonstrovány schopnosti OLED displeje PmodOLEDrgb. Celkově práce tvoří přehledný návod pro další využití barevných OLED displejů, které budou součástí systému řízeného programovatelnými hradlovými poli FPGA.

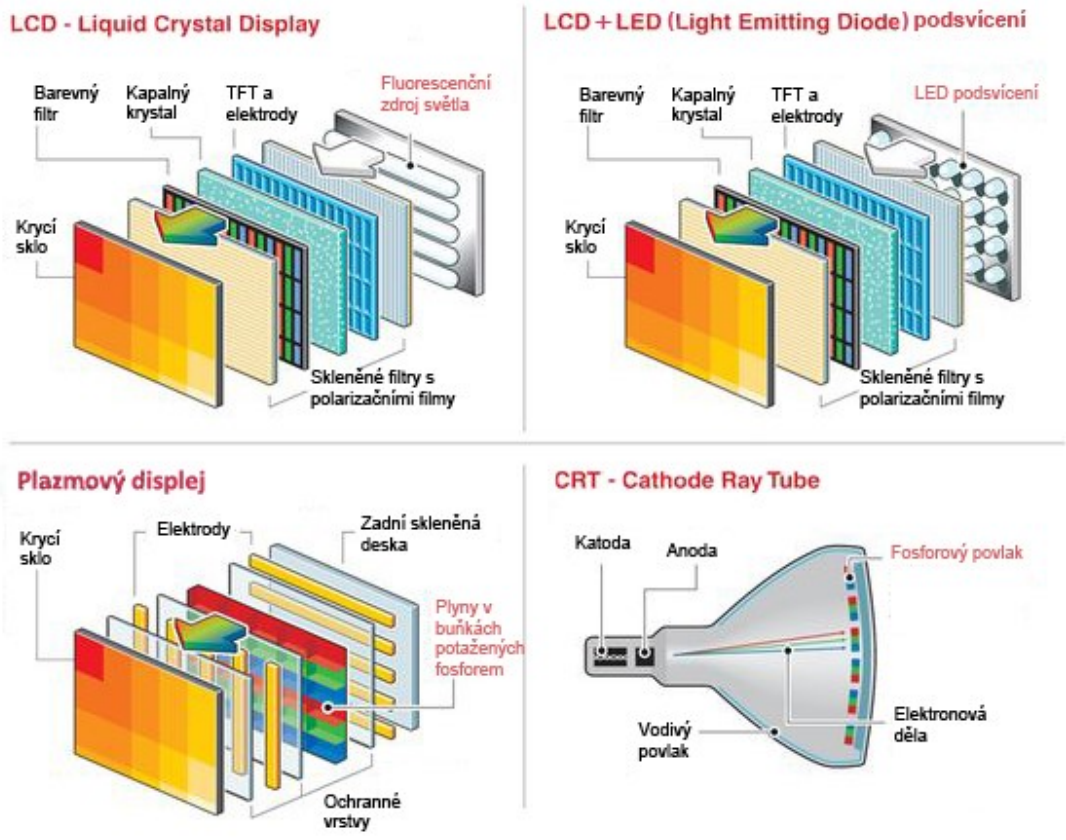
1. Historie vývoje displejů

CRT – Fungují na principu urychlovače elektronů (katodové trubice) se stínítkem. Trubice je uzavřena do vzduchoprázdné baňky. Obraz se tvoří tak, že katody emitují elektronové svazky urychlených elektronů, jeden pro každou barvu základního spektra RGB. Svazky elektronů jsou vychylovány elektromagnetickými silami pomocí cívek. Elektrony dopadají na stínící mřížku, kde jsou otvory uspořádány, aby na každý luminofor v RGB spektru dopadl příslušný svazek elektronů, který určuje barvu. Luminofory ostatních barev jsou zastíněny. Elektrony při dopadu na stínítko pokryté luminoforem se rozzáří. [1]

Plazmový displej – Podobá se OLED v tom, že vydává své vlastní světlo pro vytváření barev základního spektra tj. červenou, zelenou a modrou barvu. Jednotlivé buňky obsahují xenonové a neonové plyny, jež vyzařují světlo, jsou-li zapojeny do zdroje napětí.

LCD – Technologie LCD oproti plazmě nepracuje s buňkami, které samy o sobě září. Obrazovka LCD televizoru obsahuje podsvětlovací trubice. Ty neustále svítí, a aby bylo jejich záření rozmístěno po celé obrazovce rovnoměrně, displej tvoří i několik různých vrstev, které toto světlo odrážejí a vedou do tekutého krystalu, ten ho v různých intenzitách propouští ven.

LED – LED displej je technologie zobrazení obrazu, jež jako světelný zdroj využívá panel tvořený LED diodami. Na Obrázek 1 je LED displej spojen s některými vrstvami technologie LCD pro zkvalitnění obrazu.



Obrázek 1. Předchůdci OLED displejů

2. Technologie OLED

Zkratkou OLED neboli Organic Light-Emitting Diode se rozumí polovodičový prvek vyzařující světlo, jehož tloušťka je mezi 100 nm a 500 nm. Mají sendvičovou strukturu, tzn. mezi elektrodami (anodou a katodou) jsou dvě (v modernějších modelech tři) organické vrstvy.

Obrázek 2 - OLED displej[2]

2.1. Konstrukce OLED

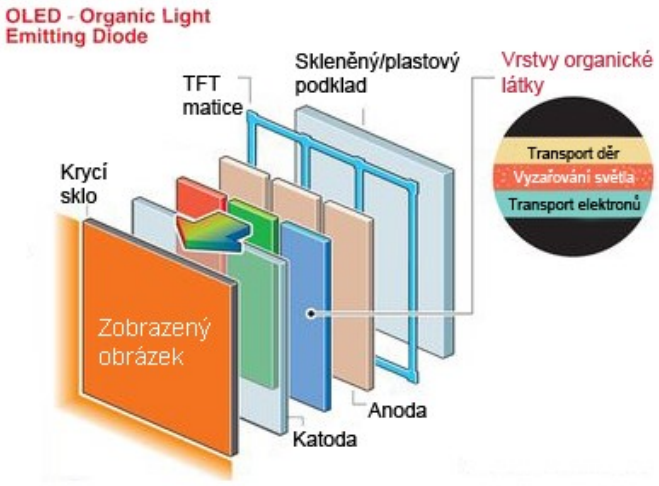
Podklad (Substrát) má podpůrnou funkci a tvoří jí sklo nebo plastová folie.

Anoda je transparentní vrstva, která odebírá elektrony a přidává „díry“, pokud diodou protéká proud.

Organická vrstva je složená z vrstev, emisní a vodivé vrstvy.

- **Vodivá vrstva** je vyrobena z organických plastových molekul, které pomáhají transportovat díry z anody. Vodivý polymer používaný v organických LED je například polyanilin. [3]
- **Emisní vrstva** vyrobená z organických plastových molekul přenáší elektrony od katody. Vrstva je vyrobena ze speciálních organických molekul, které vedou elektřinu. V této vrstvě se používají dva polymery, polyfluoren a polyfenylen, které normálně vydávají zelené a modré světlo. Zde vzniká světelné záření, prochází-li diodou proud.
- **Vodivá vrstva***, používá se v modernějších modelech pro zefektivnění přenosu elektronů od katody.

Katoda je zodpovědná za dodávání elektronů diodě, pokud zařízením protéká proud. Vrstvu většinou tvoří kov nebo také vápník, baryum, hliník a hořčík, podle toho jestli se jedná o transparentní nebo neprůhledný OLED.



Obrázek 3- Struktura OLED displeje (AMOLED)

2.2. Princip fungování OLED

Princip fungování organických světlo emitující diod pracuje na velmi podobném principu jako klasická LED.

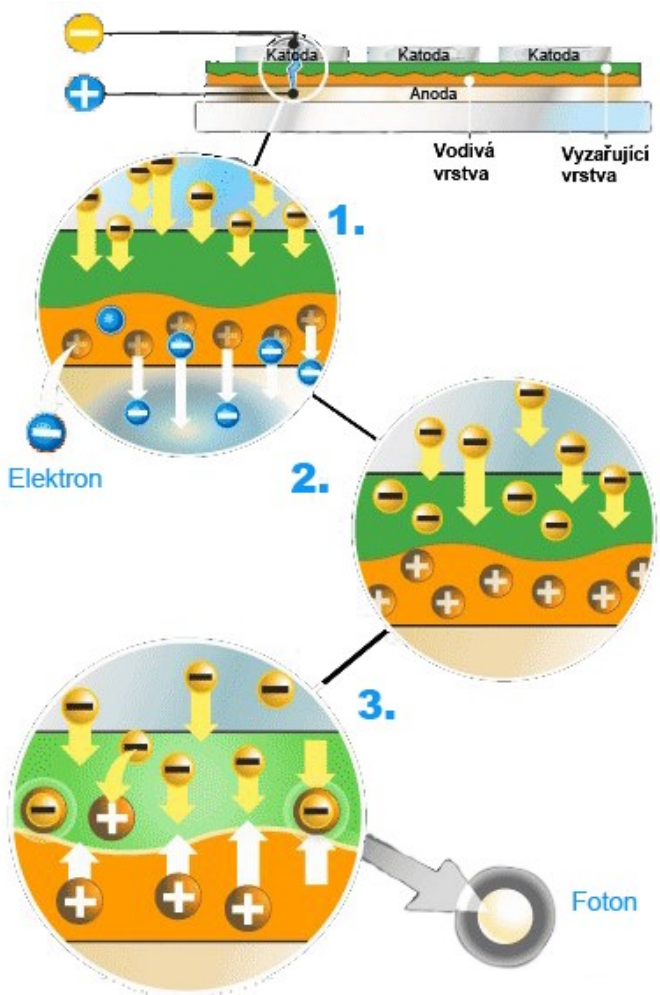
Elektrický proud z katody protéká k anodě (jako u klasické LED, jen s tím rozdílem, že mezi anodou a katodou proud prochází ještě organickou vrstvou, která je složena z vodivé a emisní vrstvy, přičemž předává elektrony do vyzařující vrstvy a odebírá je z vrstvy vodivé.

Odebírání elektronů z vodivé vrstvy zanechává díry, které potřebují být zaplněny elektrony ve vyzařující vrstvě.

Díry přeskakují do vyzařující vrstvy, dochází k rekombinaci děr generovaných anodou a elektronů vysílaných katodou a při tomto procesu uvolňují svou přebytečnou energii v podobě optického záření (světla).

Struktura organických vrstev určuje barvu vyzařovaného optického záření.

[22]



Obrázek 4 – Základní princip na jakém fungují OLED[upraveno ze 4]

2.3. Typy OLED

OLED jsou klasifikovány dle velikosti molekul tvořících organický materiál, ze kterého jsou vyrobeny. Z toho pohledu existují dvě základní technologie:

- SMOLED (Small Molecule LED), technologie používá relativně malé molekuly. Molekuly jsou nanášeny metodou vakuového napařování.
- LEP (Light Emitting Polymers), využívají velké molekuly známé pod názvem optické záření emitující polymery PLED.

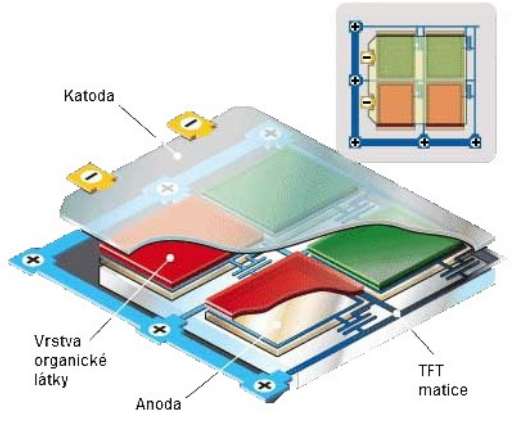
[22]

Dnes je dostupná široká škála konstrukčních typů OLED displejů, přičemž různé typy se hodí pro jiné aplikace. Nejběžnější v současné době na trhu jsou typy AMOLED a PMOLED. Jejich konstrukce je tedy dále více rozvedena a u ostatních typů jsou uvedeny pouze bazální informace.

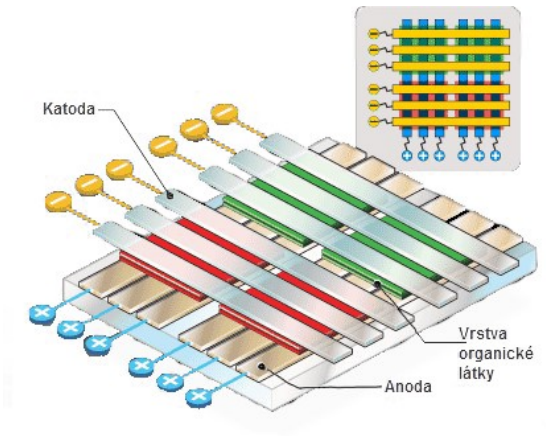
Tabulka 1. Srovnání AMOLED / PMOLED

	OLED s aktivní maticí	**OLED s pasivní maticí**
	Active-matrix OLED	*Passive-matrix OLED*
Umístění organické vrstvy	Organickou vrstvu svírají celistvé vrstvy katody a anody	Organická vrstva je mezi pásky katody a anody, které jsou položeny kolmo na sebe
Způsob řízení displeje	Anodu překrývá TFT (Thin Film Transistor) matice, která sama rozhoduje, který pixel se zapne.	Každé políčko, kde se kříží elektrody, je samostatným pixelem
Klady a zápory	Rychlejší odezva	Nejjednodušší na výrobu
	Vyžaduje méně energie ve srovnání s PMOLED	Má vyšší spotřebu, kvůli nutnosti napájet externí řídící obvody
	Vhodné pro velké obrazovky	Vhodné pro malé displeje

TFT – Tenkovrstvé tranzistory umožňují zrychlit zobrazování displeje, takže jeho obnovovací frekvence je vysoká a nedochází k chvění obrazu. TFT, které jsou v sérii s aktivní oblastí každé buňky displeje, řeší problém efektivní adresace tím, že řídí velikost proudu protékajícího buňkou, také slouží k nastavení jasu.[22]



Obrázek 6 – AMOLED [upraveno z 5]



Obrázek 5 – PMOLED[upraveno z 5]

Další typy OLED displejů jsou tyto:

TOLED - Transparentní OLED – Skládá se z průhledného podkladu, anody i katody, toto umožňuje až 70% průhlednost. Světlo září v jednom či obou směrech. Tyto typy OLED jsou užitečné pro hledí přilby, transparentní projekční plátna a brýle.

WOLED - Bílá OLED – Vydávají pouze bílé světlo a používají se při výrobě větších a účinnějších osvětlovacích systémů. Mohou nahradit zářivky, jelikož náklady na spotřebu jsou výrazně nižší. [5]

FOLED - Flexibilní OLED – Jsou vyrobeny z pružné kovové nebo plastové fólie. Výhody technologie jsou nízká hmotnost, nižší náklady na výrobu, ultra tenký pružný vzhled, čímž se snižuje hrozba rozbití.

PHOLED - Fosforeskující OLED – Pracuje v zásadě na stejném fyzikálním základu a efektu, jen využívá jiné materiály emitující vrstvy. Pracují při velmi nízkém napětí, mají dlouhou životnost a méně se zahřívají, protože jsou až čtyřkrát efektivnější (úspornější), neboť přeměňují více energie na světlo, než na teplo. [5]

SOLED – Vrstvený (stacked) OLED – Displej se skládá z vertikálně na sebe poskládaných RGB buněk TOLED. Tato technologie umožňuje zvýšení rozlišovací schopnosti displeje (všechny tři barvy jsou obsaženy v horizontálním rozměru jedné, tzn. trojnásobně vyšší rozlišitelnost) a zlepšení kvality barevného zobrazení. Barvu každé buňky displeje je možné změnit změnou proudu jednotlivými diodami.[22]

2.4. Zhodnocení a využití OLED displejů

V dnešní době je OLED technologie jedna z nejperspektivnějších na trhu. Má řádu výhod proti běžnějším technologiím (LCD, LED), z těch nejvýznamnějších je možné jmenovat tyto:

Výhody

- OLED displeje nepotřebují podsvícení, zatímco LCD displeje ho potřebují, neboť fungují na principu blokování světla, což pro OLED znamená výrazně nižší spotřebu.
- Absence externího podsvícení přináší základní a nejvýznamnější rozdíl v kvalitě obrazu. Tím, že jednotlivé body (pixely) svítí, či mohou být kompletně černé, takže obraz je ostřejší, detailnější a kontrastnější.
- Další výhodou jsou velké pozorovací úhly dosahující až 170°.
- Rychlá doba odezvy, řádově až tisíckrát kratší než u běžných LCD displejů
- OLED displeje jsou lehké, tenké (až 4 mm) a mohou být z ohebných, či průhledných materiálů, na rozdíl od LCD, což otevírá různorodé možnosti.
- Příjemnější pro oči při dlouhodobějším vystavení.

[22]

Na druhou stranu s OLED souvisí i jisté nevýhody:

Nevýhody

- Barevná vyváženost a s tím spojená životnost displeje. Látka využívaná k tvorbě modrého světla degraduje časem rychleji, než látky produkující ostatní barvy (červená, zelená). Vzhledem k ostatním bude modrá barva produkovat méně světla.
- Dále cena je zatím poměrně vysoká cena, neboť se v ní ještě odráží dosavadní drahý vývoj, což by se mělo s masovější výrobou změnit.

[22]

Využití

Vezmeme-li v potaz, že displeje tvoří nenahraditelnou součást většiny nejen lékařských přístrojů a zařízení. A vzhledem k tomu, že v současnosti grafické displeje s využitím organické vrstvy mezi elektrodami v diodě patří k nejkvalitnějším a nejperspektivnějším technologiím na trhu, tak tato kombinace nabízí obrovské zkvalitnění široké škály přístrojů, jež obsahují displej, co už není tak zřejmé, OLED panely se dají využít i jako osvětlení, které také nabízí mnohé výhody, které jsou výše rozebrány podrobněji.

Ve zdravotnictví jsou tyto výhody, které OLED přináší, v prvé řadě využity na monitorech a displejích různých zařízení, jako je například 3D systém pro endoskopická vyšetření na Obrázek 8., neboť detailnější zobrazení znamená také kvalitnější a snazší práci zdravotního personálu.

Obrázek 7- OLED Monitor[6]

Obrázek 8 - 3D head-mounted display systém [7]

OLED panely jako osvětlení

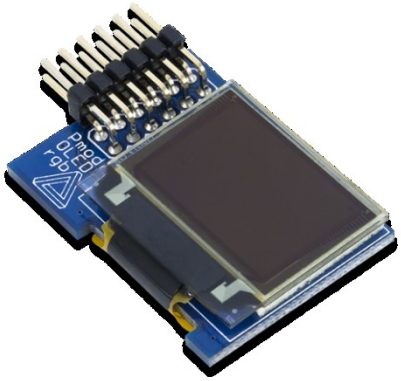
Výhody proti klasickým žárovkám a zářivkám jsou, že se jedná o levnější způsob vytvoření flexibilního osvětlení vyžadující méně energie, při lepší kvalitě osvětlení, což umožňuje nové konstrukční koncepce pro vnitřní osvětlení, které nevrhá rušivé stíny. Teplota barev odpovídá reálu a svítí to příjemnějším světlem, tím je myšleno, že nepoškozuje zrak lékaře ani pacienta při jeho dlouhodobém vystavení.[8]

Obrázek 10 - Srovnání OLED jako zdroj světla [8]

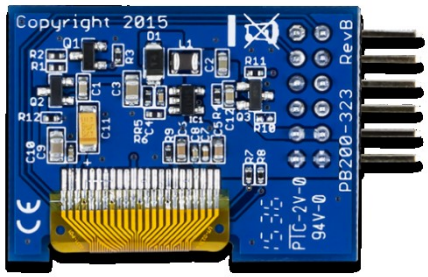
Obrázek 9 - Výhody OLED jako zdroj světla [8]

2.5. Displej Pmod OLEDrgb

K realizaci demonstrační úlohy byl vybrán barevný organický LED displej Pmod OLEDrgb od firmy Digilent.



Obrázek 12 - Pmod OLEDrgb (zepředu)[9]



Obrázek 11 - Pmod OLEDrgb (zezadu)[9]

Tabulka 2. Parametry PmodOLEDrgb[25] [9]

Vlastnost	**Hodnota**
Rozlišení obrazu	96 x 64 pixelů
Celkové rozměry prvku	25,70 x 22,20 x 1,50 mm
Aktivní oblast	20,14 x 13,42 mm
Barevné rozlišení	16 bitové
Typ displeje	PMOLED
Velikost pixelu	0,05 x 0,19 mm
Řadič displeje	SSD1131
Hmotnost	1,8 g
Způsob komunikace	12 pinový Pmod s rozhraním SPI
Doporučené napájecí napětí	3,3 V
Životnost	10 000 (20 000) hodin

Displej je 16 bitový, což znamená, že je schopen vytvořit 65 536 různých odstínů barev. Organický LED modul od firmy Digilent ovládá řadič displeje Solomon Syntech SSD1331, slouží pro příjem informací z hostitelské desky a zobrazování požadovaných informací na OLED obrazovce. [9]

2.6. Řadič displeje

2.6.1. Popis řadiče

SSD1331 je jednočipový řadič OLED displeje s 288 segmenty, podporující 96 x 64 RGB pixelový displej. Jeden pixel obsahuje 3 segmenty, pro každou barvu RGB jeden, jejich kombinací se skládá výsledná barva pixelu. Tento čip je určen pro běžný typ katody OLED panelu.

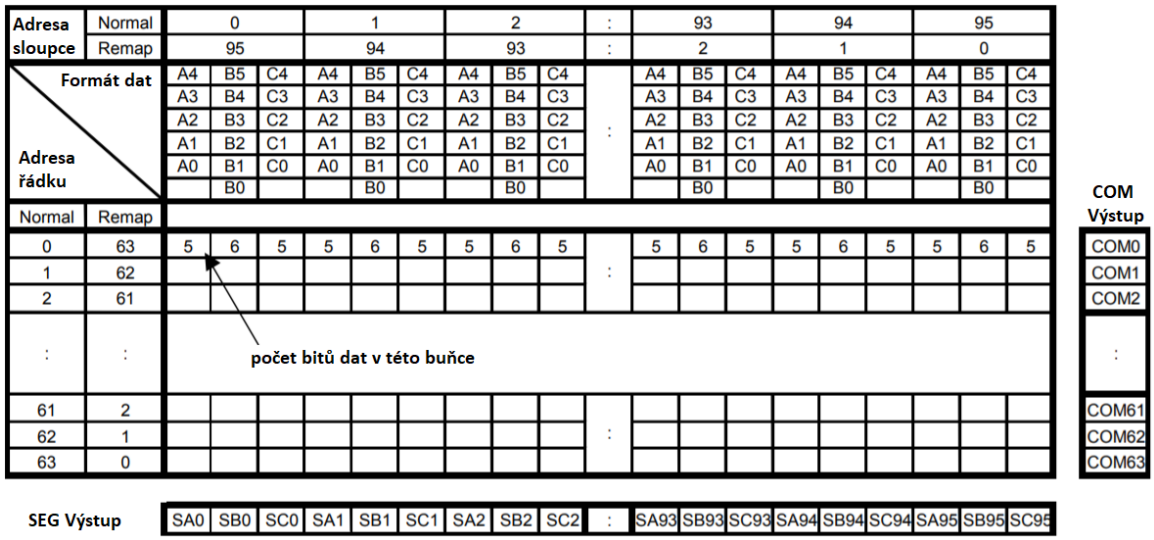
SSD1331 zobrazuje data ze zabudované GDDRAM (Graphic Display Random-Access Memory). Podporuje přenos dat a příkazů od nadřízeného procesoru přes 8080/6800 paralelní rozhraní nebo SPI (Serial Peripheral Interface). Umožňuje regulovat kontrast displeje v 256 úrovních a má naprogramovanou tabulku stupňů šedi.

Řadič SSD1331 společnosti Digilent je dodáván s řadou předdefinovaných grafických funkcí, které uživatel může využít (například zobrazení znaků z ASCII tabulky a nastavení umístění kurzoru).

[10] [9]

2.6.2. Organizace paměti GDDRAM

GDDRAM je bitmapová statická RAM, neboli paměť RAM grafického displeje, která obsahuje konfiguraci, která má být zobrazena. Velikost RAM je 96 x 64 x 16 bitů. Přemapování jak segmentů, tak celých řádků může být zvoleno i softwarem. Každý pixel obsahuje 16 bitová data. Tři sub-pixely pro každou barvu A (R), B (G) a C (B) každý má buďto 5 nebo 6 bitů. Uspořádání dat pixelů v GDDRAM je ukázán na Obrázek 13.



Obrázek 13. 65k Barevná hloubka GDDRAM struktury[upraveno z 10]

2.6.3. Stupnice úrovní šedi

Řadič umožňuje řídit jas jednotlivých pixelů pomocí 16 bitů, které jsou uloženy na odpovídajích místech v paměti. Tothoto řízení dosahuje pomocí PWM modulace napájecího signálu pro jednotlivé body. Proto je možné nastavit si tabulku šedi, v níž dojde k přiřazení 64 hodnot, které ma řadič možnost použít v paměti k některým hodnotám ze stupnice šedi. V Tabulka 3. je výchozí nastavení této převodní tabulky. Efekt úrovně šedé barvy je regulován šířkou pulzů segmentu Čím je pulz širší, tím je pixel světlejší. Šířka pulzů může být nastavena příkazy softwaru. Sub-pixely barvy A (červená) a C (modrá) mají jen 5 bitů, reprezentovaných 32 úrovněmi stupně šedi. Kdežto barva B (zelená) má velikost 6 bitů, reprezentovaných 64 úrovněmi šedi.

Tabulka 3. Vztah mezi obsahem paměti GDDRAM a výchozí tabulkou stupnice šedi [10]

GDDRAM data (5 bitů)	GDDRAM data (6 bitů)	Tabulka stupnice šedi	Výchozí nastavení jasu
00000	000000	GS0	0
-	000001	GS1	1
00001	000010	GS2	3
-	000011	GS3	5
:	:	:	:
:	:	:	:
-	111101	GS61	121
11111	111110	GS62	123
-	111111	GS63	125

Čtyři fáze řízení jasu OLED pixelu:

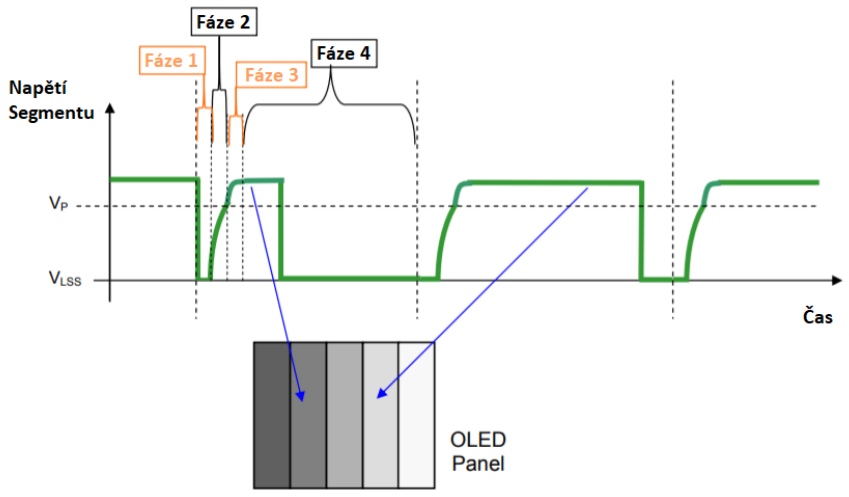
Fáze 1: Ovladač segmentu znovu nastaví výchozí hodnoty pixelu na V_{LSS}(zemnící proud), aby se odstranil předchozí obsah uložený v parazitní kapacitě.

Fáze 2: Provede se první předběžné nabití. Pixel je řízen tak, aby se dostal na odpovídající úroveň napětí V_P z V_{LSS}.

Fáze 3: OLED pixel je přiváděn k cílovému řídícímu napětí prostřednictvím druhého předběžného nabíjení. To může řídit rychlost nabíjení.

Fáze 4: Proudový zdroj ovladače segmentu poskytuje konstantní proud přiváděný k pixelu. Ovladač používá pulzně šířkovou modulaci (PWM – Pulse Width Modulation) k ovládání stupnice šedi každého pixelu jednotlivě. Širší impulsy mají za následek jasnější body a naopak tenčí impulsy tvoří tmavší body. Toto je znázorněno na následujícím Obrázek 14.

[9][10]



Obrázek 14. Úrovni šedi řízené PWM v segmentech [upraveno z 10]

3. Programovatelné obvody

Číslicové systémy je možné vytvořit několika způsoby a to buďto s využitím ASIC, DSP CPU nebo FPGA. Dnes se vyrábějí převážně ve formě integrovaných logických obvodů, přičemž z ekonomického hlediska hraje hlavní roli masová výroba. Takže se dbá na to, aby součástky byly co nejuniverzálnější. Z toho plyne, že musí zvládat pracovat v různých režimech činnosti, aby bylo možné pro každou aplikaci vybrat ten nejvhodnější.

Tabulka 4. Možnosti realizace číslicových systémů [21]

Technologie vlastnosti	**ASIC**	**DSP CPU**	**FPGA**
Charakter návrhu	Všechny druhy návrhu – asynchronní, synchronní, nízkopříkonový, analogové obvody.	Lze realizovat vše, co lze naprogramovat. Vlastnosti systému jsou určeny především architekturou použitého CPU. DSP procesory jsou určeny pro signálové aplikace, mají speciální funkční jednotky pro zjednodušení a zrychlení výpočtů.	Plně synchronní návrh, nedoporučuje se používat asynchronních technik návrhu a je třeba pečlivě řešit případné přepínání či hradlování hodin v obvodu.
Cena návrhu	Drahý návrh, vysoké fixní náklady.	Nejlevnější návrh.	Nízké fixní náklady.
Náročnost	Profesionální pracoviště.	Jednoduché programování	I méně odborně fundovaná pracoviště.
Dosažitelný výpočetní výkon	Mohutná paralelizace, multiplexování bloků, vysoký výpočetní výkon.	Předem dané množství výkonných jednotek, často nemožnost všechny jednotky vytížit tokem instrukcí.	Mohutná paralelizace, multiplexování bloků, vysoký výpočetní výkon.

3.1. Programovatelná hradlová pole

Programovatelná hradlová pole (Field Programmable Gate Array – FPGA) jsou integrované obvody, které jsou tvořeny velkým množstvím buněk uspořádaných do matic, jejíchž funkci lze jednoduše měnit přeprogramováním. FPGA patří mezi nejvýkonnější představitele programovatelných logických součástek (Programmable logic device – PLD) na trhu. Obsahují na čipu pravidelnou strukturu konfigurovatelných logických bloků CLB. Propojením logických bloků lze vytvářet nová komplexnější zapojení, k jejíchž realizaci by bylo zapotřebí mnoho různých obvodů. Vstupní a výstupní signály výsledného zapojení jsou pak připojeny k vývodům hradlových polí. V současnosti FPGA tvoří konkurenta zákaznickým integrovaným obvodům (ASIC). Klíčovou výhodou oproti nim má FPGA v programovatelnosti, což znamená, že konfigurace obvodu FPGA je na uživateli, na rozdíl od ASCI, kde je obvod již předem naprogramován. Dalšími výhodami jsou vysoký výpočetní výkon, rekonfigurovatelnost návrhu, podpora vysokorychlostní komunikace, možnost rychlého číslicového zpracování signálu a podpora multimediálních aplikací. Mezi hlavní výrobce FPGA patří Altera, Lattice, Actel, Cypress a především Xilinx. [13] [21]

3.1.1. **Základní prvky FPGA**

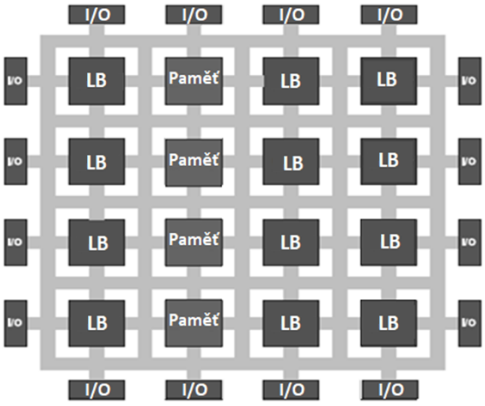
Logické bloky (CLB – Configurable Logic Block) jsou základní stavební prvky vytvářené logiky.

LB mohou obsahovat:

- Každý blok je tvořen posuvným registrem, zvaným LUT (Look Up Table), to je bitová paměť schopná realizovat logické funkce.
- Pomocný multiplexor umožňuje propojovat LUT s výstupními registry, které reagují na hranu hodinového signálu.
- Součást tvoří specializované bloky například paměť RAM, procesorová jádra a další.

IOB (Input/Output Block) neboli vstupně/výstupní bloky tvoří rozhraní mezi vnějšími vývody součástky a signály vnitřní logiky, přičemž každý může být konfigurován jako vstupní, výstupní nebo obousměrný.

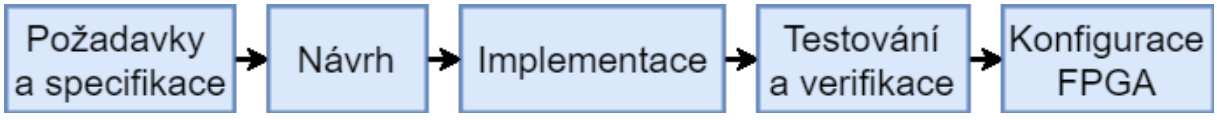
Síť lokálních a globálních propojovacích prostředků slouží k propojení signálů mezi jednotlivými bloky LB a mezi LB a IOB



Obrázek 15. Architektura FPGA

3.2. Postup při návrhu FPGA

Prvním krokem by měl být podrobný popis požadavků na funkci. Systémy FPGA se programují nejprve na úrovni hardwaru. Z toho plyne, že se při programování přímo řídí i propojovaní jednotlivých funkčních bloků a konfigurují výše zmíněné logické bloky. To znamená, že výsledným produktem je naprogramovaná struktura, jež pracuje jako paralelní systém s určitým hodinovým cyklem v řádu až stovek MHz. Druhým krokem bývá návrh systému a popis zapojení realizovaného v FPGA. Jakmile je zapojení funkční přistupuje se k syntéze, během té se HDL kód převede do netlistu (popisu elementárních bloků). Následuje fáze implementace, první se provede mapování do konkrétní komponenty FPGA, kdy se logické prvky nahradí těmi v konkrétní architektuře. Kombinační logika se namapuje do LUT, sekvenční do DKO. Poté se vytvořený design rozmístí do struktury čipu. V následujícím kroku se vytvoří binární konfigurační soubor (.bit). Po úspěšné implementaci do FPGA zbývá provést ladění a ověření funkčnosti celého zapojení v reálné aplikaci. [12]



Obrázek 16. Diagram postupu návrhu

Návrh systému

K tomu aby vývojová deska pracovala dle potřeb uživatele, musí se nejdříve specifikovat požadavky na vytvářený logický obvod. Poté je uživatel převede do zdrojového kódu, specifikace funkce, zachycení návrhu, možná verifikace na úrovni RTL, implementace a výsledkem je konfigurace obvodu na vývojové desce. [21]

Zachycení a kódování návrhu

Jsou kroky, při kterých se přenáší představa o funkci, struktuře budoucího obvodu a o požadavcích specifikace do počítačem zpracovatelné formy popisu požadované číslicové funkce. Zachycení návrhu lze provádět na různých úrovních abstrakce. *Schématicky*, kdy se navrhuje přímo kreslením schématu budoucího obvodu. Návrh je sice srozumitelný, ale kreslené schéma je obvykle specifické pro daný obvod, takže nastane problém v případě konverze do jiného obvodu. Další možností jsou dnes standardně používané *meziregistrové přenosy,* tzn. RTL (Register Transfer Level). Popis je realizován v programovacím jazyce HDL (Hardware Description Language), který definuje jednotlivé struktury a jejich propojení. Nejvyšší úroveň abstrakce žádá *algoritmické* zachycení, které spočívá v převedení RTL kódu výsledného systému (datových cest i řídících bloků) do aritmetických jednotek.[21]

RTL simulace

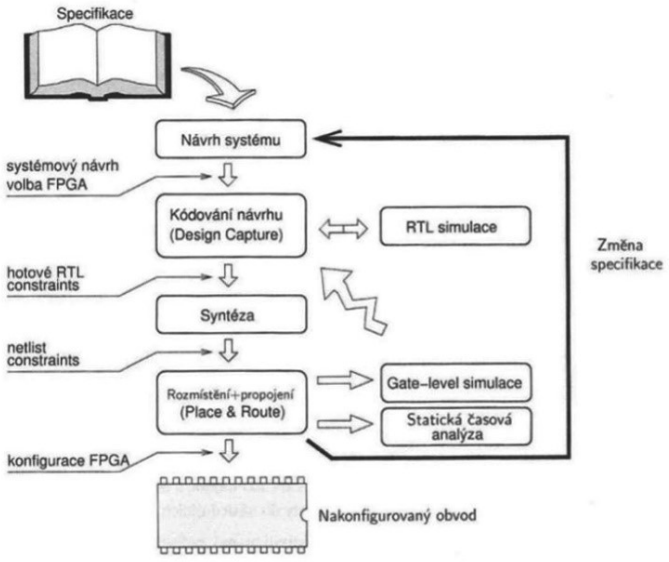
Při tvorbě designu je dobré pamatovat na verifikaci vznikajícího řešení. Verifikace ověřuje pomocí simulací, správnost a dodržení časových parametrů prvků. RTL simulace je vhodná pro ověření správné funkce RTL HDL popisu a reakce obvodu podle očekávání. [21]

Implementace

Během implementace probíhá konverze VHDL popisu do konfiguračních dat pro FPGA. Prvním krokem implementace je mapování konkrétní součástky FPGA a pak následuje rozmístění a propojení se strukturou čipu, tím se vytvořený design rozmísťuje do FPGA (rozložením pinů, umístěním bloků FPGA, atd.). Výsledkem implementace je opět netlist, ovšem takový, který je závislý na konkrétní součástce.[21]

Statická časová analýza

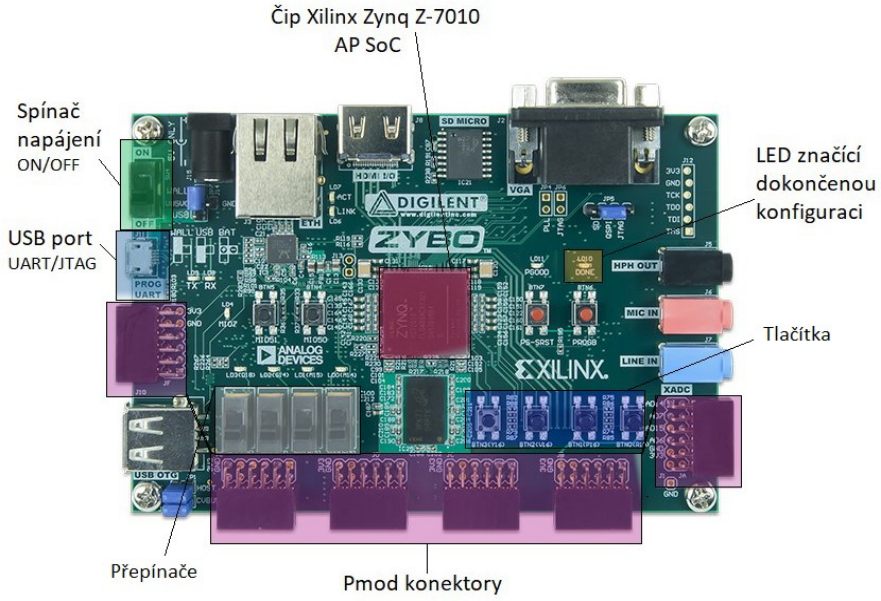
STA (Static Timing Analysis) nástroj analyzuje všechny logické cesty v obvodu a ověřuje, zdali je vytvořené zapojení schopno plnit časové nároky (tj. pracovní frekvence, zpoždění mezi IN/OUT). [21]



Obrázek 17. Model návrhového procesu [21]

3.3. Popis desky ZYBO

Vývojová deska ZYBO (ZYnq BOard) poskytuje základní platformu pro návrhy programovatelné logiky zamřené na architekturu Xilinx Zynq Z-7010 AP SoC, která integruje dvou jádrový ARM Cortex-A9 procesor s FPGA logikou. Je vybavena množstvím vstupně/výstupních periferních zařízení, krystalovým oscilátorem a obvodem FPGA, obsahujícím 28 000 logických buněk. Deska ZYBO je kompatibilní s Xilinx Vivado Design Suite stejně jako se sadou nástrojů SDK.[14]



Obrázek 18. Deska ZYBO [upraveno z 13]

Mezi dostupné komponenty patří GPIO (6 tlačítek, 5 LED a 4 tahové spínače), 6 pmod konektorů, audio kodek se sluchátkovým výstupem, jackem pro mikrofon, MicroSD slot, HDMI Port, VGA výstupní port, Ethernet, UART na USB převodníky (UART se používá podle kontextu aplikace v SoC pro odesílání a přijímání dat z případně do externího zařízení. JTAG se používá k ověření logiky obvodu a testovacího zařízení.).

3.4. Architektura ZYNQ 7000 Development Board

Vestavěný počítačový systém, má svou řídící jednotku zabudovanou do zařízení, jež řídí. Typické vestavěné systémy jsou většinou složeny primárně z částí, nezbytných pro funkci, pro kterou byly navrženy. Konkrétně systémy z řady Xilinx Zynq se soustředí na vývoj prototypových řešení pomocí vývojových desek a dalších zařízení. [16,17]

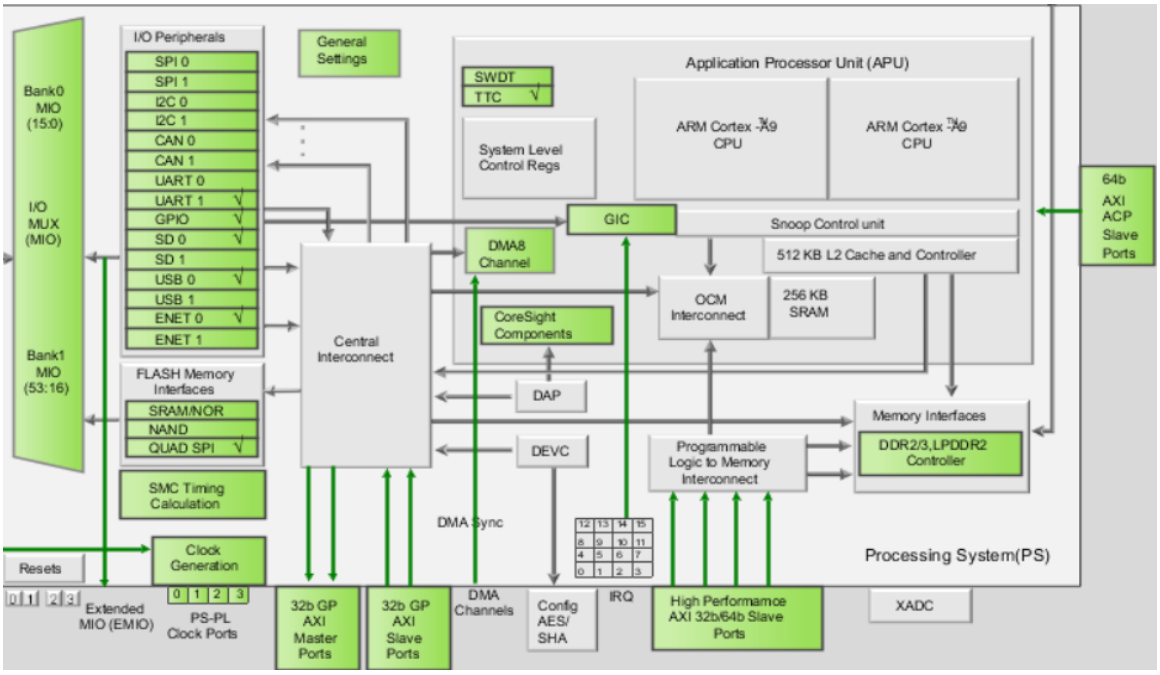
Elementárním znakem architektury tohoto vestavěných systémů je kombinace dvou navzájem propojených částí, a to procesorového systému (Processing systém – PS) a programovatelné logiky (Programming logic – PL). Na Obrázku je vyobrazen zjednodušený model architektury a propojení jednotlivých částí. [16,17]

3.4.1. Programovatelná logika

Programovatelná logika (PL) je stěžejní částí architektury, hlavně protože je nezávislá na PS a má vysokou výpočetní rychlost na rozdíl od PS. Klíčovou částí programovatelné logiky je programovatelné hradlové pole (FPGA), kde je možné vytvářet paměťové bloky, aritmetické a logické operace a hlavně vytvářet propojení jednotlivých komponent PL. Pro vytváření programů pro programovatelnou logiku se využívá nespočet technologií, jež spojuje typ výstupu, jímž je soubor integrující data pro FPGA. [15]

3.4.2. **Procesorový systém**

Procesorem řízená část architektury. Hlavní částí systému je dvou jádrový procesor ARM Cortex-A9. Tím pádem je možné používat operační systém a s tím související uživatelské programy. Procesorový systém je stěžejní částí při návrhu a implementaci systému pro ovládání prvků. [15]



Obrázek 19. Blokové schéma architektury Xilinx Zynq 7000 z programu Vivado

APU (Aplikační procesorová jednotka) se skládá především ze dvou procesorových jader ARM Cortex-A9 (CPU). Každé z nich má přidružené výpočetní jednotky, jednotky pro správu paměti (MMU) a vyrovnávací paměti 1 úrovně. SCU (Snoop Control Unit), má za úlohu udržovat koherenci dat v mezipamětích procesorů Cortex-A9 a tvoří most mezi mezipamětmi 2 úrovně, pamětí OCM (On Chip Memory) a centrální procesorovou jednotkou (CPU).

Komunikace mezi PS a externím rozhraním je dosažena především prostřednictvím **rozhraní MIO** (Multiplexed Input/Output), které poskytuje 54 flexibilních pinů, tzn. mapování mezi periferiemi a piny lze definovat dle potřeby.

Dostupné **I/O Periferie** zahrnují standardní komunikační rozhraní, SD, Ethernet, USB, UART a GPIO (General Purpose Input/Output), který lze použít pro různé účely včetně ovládání jednoduchých tlačítek, spínačů nebo LED.

PL přijímá z PS čtyři oddělené **hodinové signály** (0-3), které mohou mít různou frekvenci, do PS se dostanou z generátoru hodinového signálu, na který je přiváděn signál z krystalového oscilátoru.

Rozhranní slouží k přesouvání dat mezi CPU a zbytkem FPGA nebo naopak. AXI GP (General Purpose) Master nebo Slave je 32 bitová sběrnice, která je vhodná pro komunikaci mezi PL a PS. Existují celkem čtyři obecná uživatelská rozhraní s tím, že pro dva je PS v režimu master a pro zbylé dva je PL v režimu master. ACP (Accelerator Coherency Port) tvoří asynchronní spojení mezi PL a SCU v APU se 64bitovou sběrnicí. Tento port se využívá k dosáhnutí koherence mezi prvky PL a mezipaměti APU. PL je v režimu master. HPP (High Performace Ports) čtyři rozhraní pro velký výkon zahrnují buffer FIFO pro čtení a zápis. Slouží pro vysokorychlostní komunikaci mezi PL a paměťovými prvky PS. Šířka dat je 32 nebo 64 bitů a PL je v režimu master u všech 4 rozhraní.

[15]

3.5. SPI rozhraní

Datová komunikace se zprostředkovává různými způsoby. Může to být paralelní nebo sériová, synchronní nebo asynchronní komunikace. SPI, je druh sériové komunikace, jež provádí komunikaci mezi integrovanými obvody. SPI rozhraní může spojovat více obvodů pomocí řídícího vodiče. Jedno zařízení je v režimu master a druhý obvod je v režimu slave. Komunikace probíhá prostřednictvím 4 vodičů, avšak jednotlivá zařízení se mohou lišit uspořádáním vzhledem ke své funkci.

Základní druhy vodičů SPI:

MOSI (Master out, Slave In) – Datový výstup zařízení v režimu master připojený ke všem zařízením typu slave.

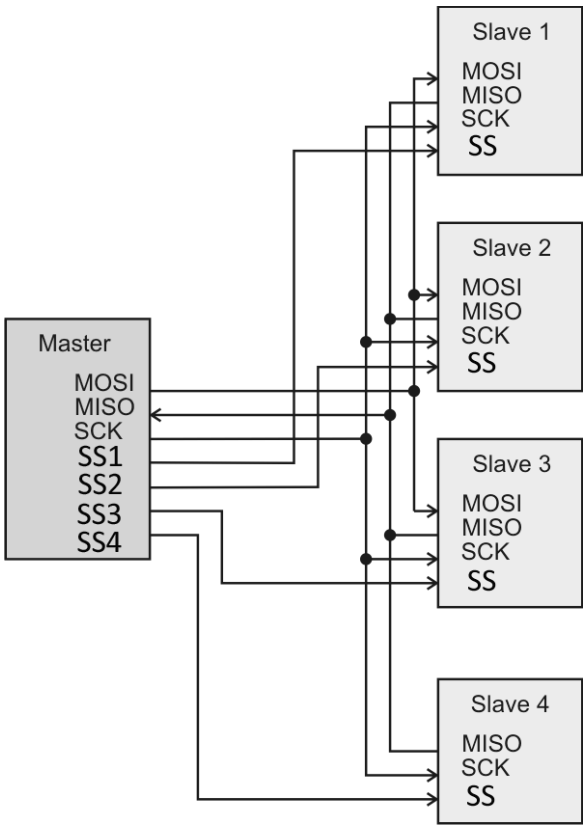
MISO (Master in, Slave out) – Výstup ze zařízení slave a vstup do zařízení typu master.

SCLK (Seriál Clock) – Hodinový signál, připojený na všechna zařízení v režimu slave.

SS (Slave Select) – Vybírá zařízení, se kterým bude v danou chvíli komunikovat.

V jeden moment vždy jeden master komunikuje s jedním slave zařízením. Oba obvody obsahují posuvné registry, které jsou posouvány generovaným SCLK z master zařízení, jakmile je registr plný dojde k jeho překlopení do paralelně zapojeného záchytného registru, kde se dále zpracovává.

[24]



Obrázek 20. SPI rozhraní [upraveno z 23]

4. Demonstrační úloha

4.1. Návrh demonstrační úlohy

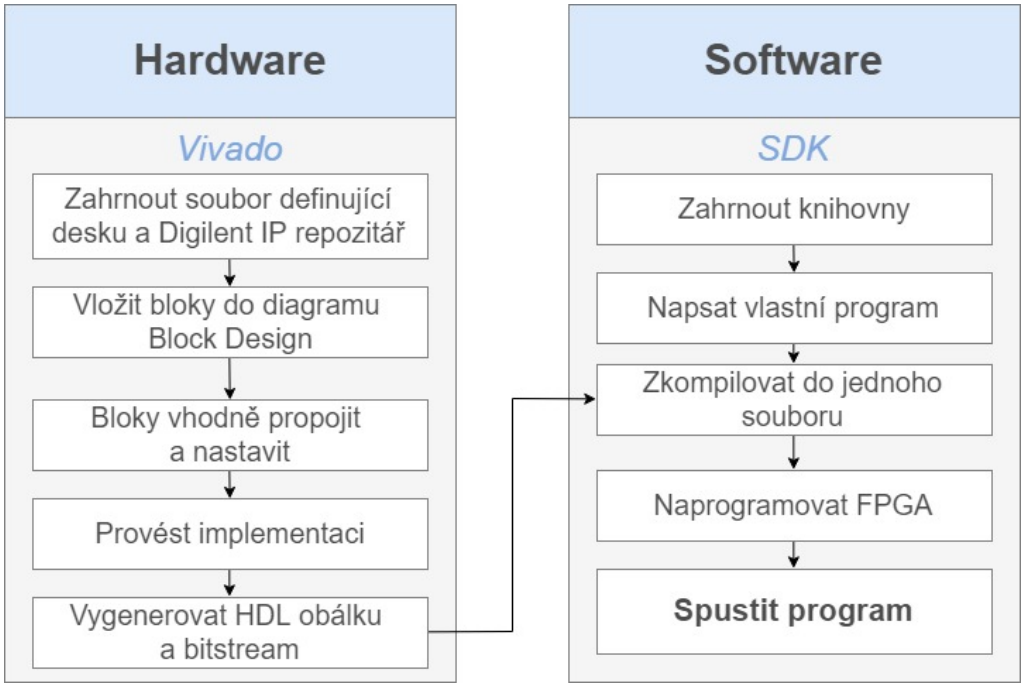
Vivado

K vývoji aplikací pro zařízení Zynq-7000, konkrétně pro část obsahující hradlová pole, slouží vývojové prostředí Vivado Design Suite od firmy Xilinx, s jehož pomocí je možné vytvořit vše potřebné k naprogramování hradlových polí (FPGA), od napsání programu v jazyce VHDL nebo Verilog, přes syntézu, implementaci až po vytvoření bitstreamu. Obsahuje také editor IP bloků a simulátor pro ověření funkčnosti. Výhodou je snadnější tvorba programu, než čistě v jazyce VHDL.

SDK

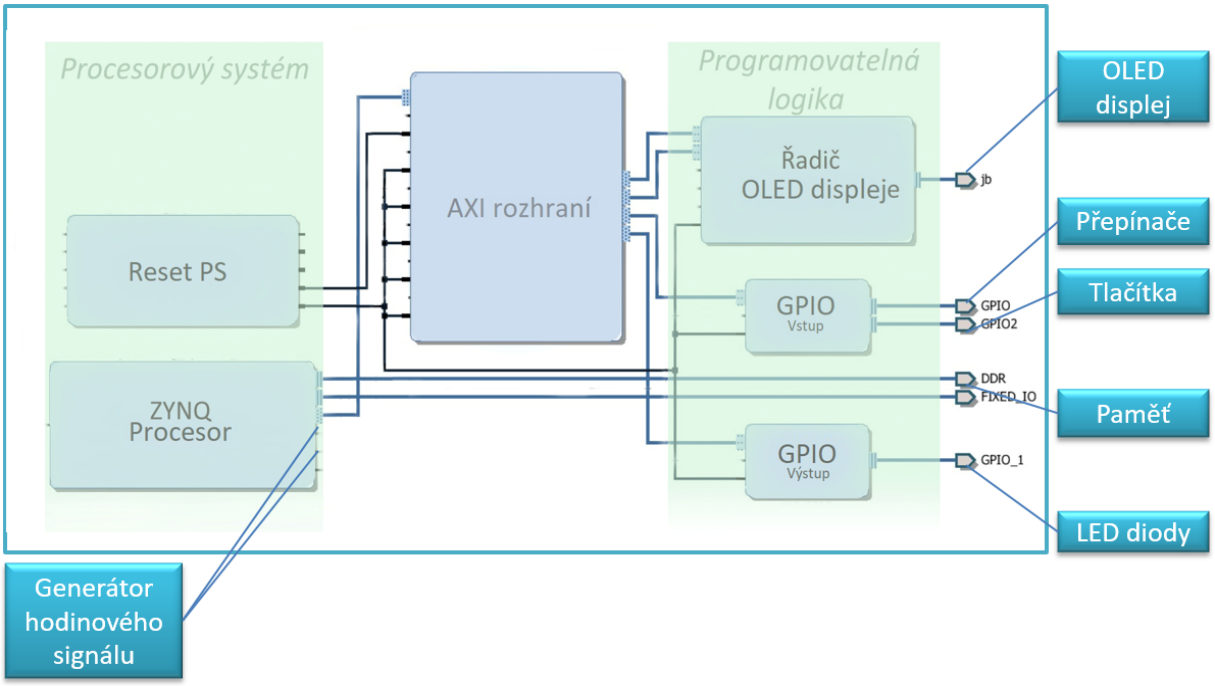
Druhý nezbytný program k vývoji projektu pro SoC Zynq, je prostředí k tvorbě aplikací, kterými jsou určeny procesorové části tohoto zařízení. Kromě toho se prostřednictvím SDK provádí kompilací výsledného bitstreamu, který vznikl v návrhovém prostředí Vivado a programu ze SDK konfigurace FPGA a následné spuštění programu.

Aby mohla být úloha vytvořena, je nutné provést tyto kroky:



Obrázek 21. Blokové schéma postupu vytváření projektu

4.1.1. Návrh HW demonstrační úlohy



Obrázek 22. Blokové schéma návrhu hardwaru systému demonstrační úlohy

Reset PS – Blok reset procesorového systému je zodpovědný za dvě funkce, pomocí dvou různých signálů. Jeden signál vyvolá návrat výchozího nastavení propojení bloků a druhý resetuje periferie.

Procesor – Modul Processing Systém 7, představuje procesorový systém Zynq®-7000 All Programmable SoC pro programovatelnou logiku a vnější logiku desky. Blok je připojen na generátor hodinového signálu, který zprostředkovává hodinový signál z krystalového oscilátoru na desce. Bližší informace k tomuto bloku se nachází v kapitole 3.4.

AXI Rozhraní – AXI Interconnect, který je nutný k propojení jednotlivých bloků pomocí AXI sběrnice. Tedy tvoří jakýsi komunikační most mezi jedním nebo více zařízeními v režimu master a s jedním nebo více zařízeními v režimu slave. Je určen pouze pro (memory-mapped) přenosy mapované pamětí. Přenosy typu stream se nedají použít.

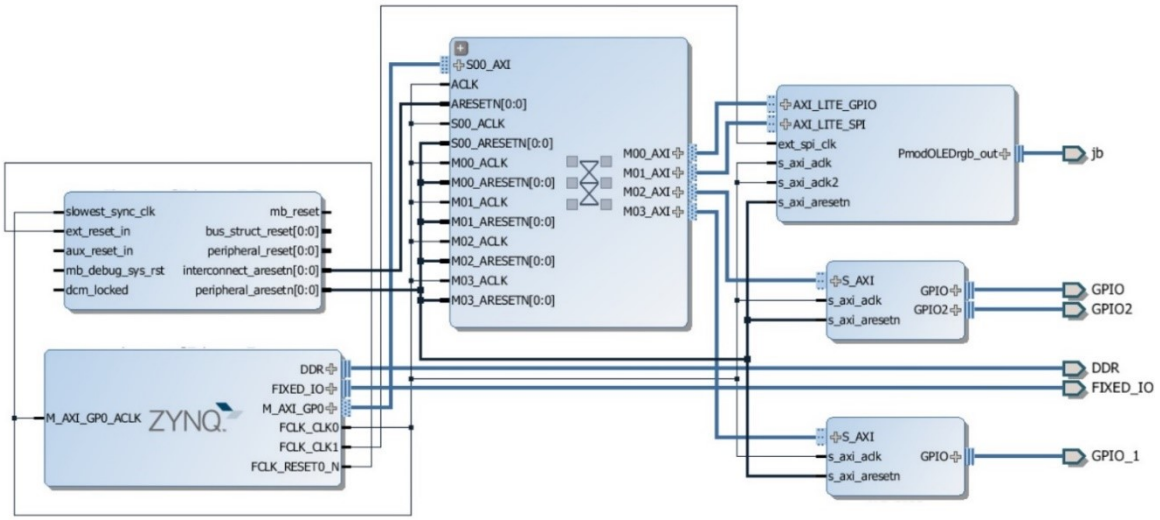
Řadič OLED displeje – Zprostředkovává komunikaci mezi procesorem a displejem, který je připojen na periferii přes konektor pmod (jb). Jeho funkcí se zabývá blíže kapitola 2.6.

GPIO – GPIO (General Purpose Input/Ouput) moduly nemají předdefinovanou funkci, mohou ovládat tlačítka, hdmi hpd (hot plug detect), hdmi výstup, LED diody nebo spínače na desce.

Na návrhu demonstrační úlohy (Obrázek 22.) je jeden GPIO blok označen jako vstup, neboť na něj jsou na periferii připojeny tlačítka a spínače, jejích kanály v GPIO modulu jsou v SW části nakonfigurovány jako vstup. Druhý GPIO modul je označen jako výstup, neboť ovládá funkci LED diod, které jsou nakonfigurovány jako výstup, tedy budou svítit podle toho, který spínač bude vybrán, tímto se dále zabývá kapitola 0.

4.2. Tvorba hardwaru

Podrobný a přehledný návod, jak vytvořit takový to projekt lze najít na konci práce v přiloženém návodu. Ve zkratce po založení nového projektu a nového Design Block diagramu je nutné vložit a graficky poskládat jednotlivé moduly, správně nastavit parametry jednotlivých bloků a vhodně je propojit. Z vytvořeného schématu se dá automaticky vygenerovat HDL wrapper, který toto schéma popisuje jazykem VHDL nebo Verilog, to závisí na tom, jaký jazyk byl zvolen při tvorbě nového projektu. Vytvořený HDL wrapper obsahuje definici vnějších vstupů a výstupů navržené aplikace. O úroveň níže je soubor desing_1.vhd, jež obsahuje volání všech použitých komponent při návrhu a definuje vnitřní signály, které se zde vyskytují a nakonec jsou jednotlivé signály připojeny ke komponentům. Na nejnižší úrovni jsou VHDL popisy jednotlivých komponent. [11]



Obrázek 23. Blokové schéma hardwaru svstému

4.3. Tvorba softwaru

Opět detailní postup, jak pracovat v SDK se nachází v příloze 2. V SDK se nejprve založí nová aplikace, kde je nutné vytvořit soubor main. c, ve kterém bude hlavní program pro procesor, zde musí být zahrnuty hlavičkové soubory (knihovny), jejichž parametry a funkce jsou dále v programu využity. [11]

Obsah knihoven:

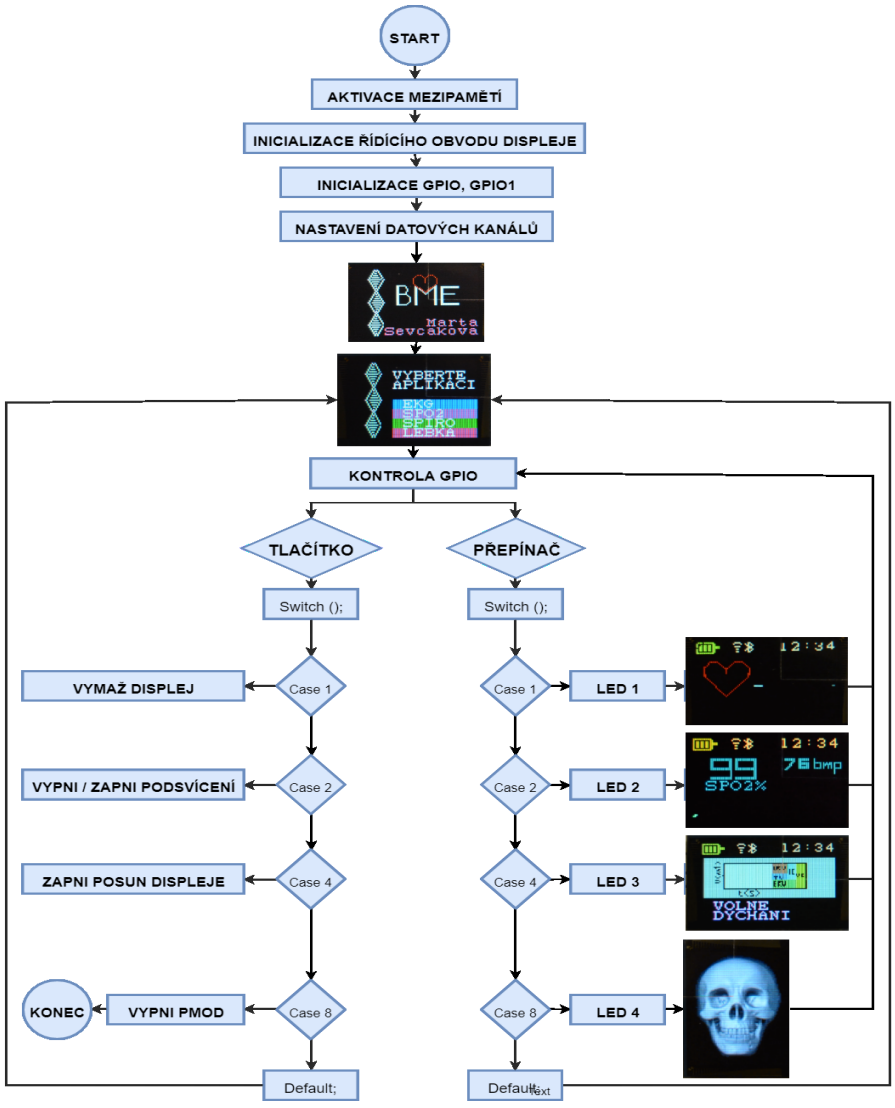
V souboru *xparameters.h* se skrývají definice všech adres, které jsou využívány pro jednotlivé komponenty připojené k AXI sběrnici. Dále *xil_cache.h* obsahuje například funkce inicializující mezipaměti. Knihovna *xgpio.h* slouží k ovládání GPIO.

Znaky abecedy a další jsou obsaženy v knihovně *ChrFont0.h*. *PmodOLEDrgb.h* je dále detailně popsána, stejně jako *xil.io.h*, *xgpio.h* a *xspi.h.* Dále *bitmap.h* obsahuje vytvořené konstanty, které zastupují bitové mapy obrázků zobrazovaných na displej funkcí *OLEDrgb_DrawBitmap* (dále popsáno v kapitole 4.4.3.).

A nakonec pro přehlednost programu byla vytvořena knihovna *pictures.h*, jež obsahuje veškeré statické části aplikací této konkrétní demonstrační úlohy. Po napsání hlavního programu se provádí kompilace bitstreamu a souboru main.c z SDK, zkompilovaným souborem se nakonfiguruje FPGA a program se spustí příkazem *Run*.

Hlavní program

Hlavní program krom knihoven obsahuje kód napsaný v jazyce C, jehož velmi zjednodušená struktura je na Obrázek 24. Struktury SW demonstrační úlohy Nejprve se provede aktivace instrukcí a datové mezipaměti (*Xil_ICacheEnable, Xil_DCacheEnable*) poté inicializace řídícího obvodu displeje (*OLEDrgb_begin*), GPIO a GPIO1 (*XGpio_Initialize*). Následně funkce *XGpio_SetDataDirection* každému kanálu modulu GPIO / GPIO1 přiřadí směr dat, tedy zdali bude přijímat, či vysílat data. Další část tvoří nekonečná smyčka while, která obsahuje kontrolu GPIO, která ověřuje zdali adresy jednotlivých tlačítek, spínačů a LED, jež jsou připojeny na kanály GPIO, mají stále stejnou hodnotu, tzn. jestli se nezměnil spínač nebo nebylo stlačeno tlačítko. Pokud nic z toho neproběhlo proběhne *default*. Pokud byl vybrán spínač, tak funkce *Switch* vybere případ a uskuteční se zapnutí LED u příslušného spínače a provede se demonstrace jedné z animací (EKG, SpO2, SPIRO nebo LEBKA). Ukázka programu těchto vizualizací je v dalé v textu rozebrána. Pokud se v průběhu výše zmíněných aplikaci zmáčkne tlačítko zavolají se jednotlivé funkce pro vymazání obsahu displeje, vypnutí a zapnutí podsvícení, posun displeje nebo vypnutí pmod, což program ukončí, deaktivuje mezipaměti a vypne zařízení.

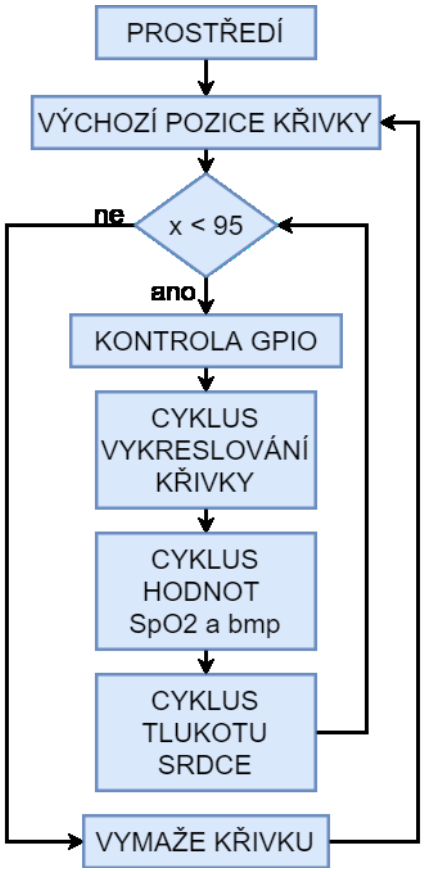


Obrázek 24. Struktury SW demonstrační úlohy

Příklad průběhu vizualizací

Většina vizualizací má podobnou strukturu jako je na Obrázek 25., proto pouze k nastínění průběhu bude blíže popsána jen pletysmografická animace, která se provede, jakmile uživatel zvolí druhý spínač, u kterého se rozsvítí příslušná LED dioda.

Nejprve se vykreslí prostředí aplikace pro tento účel vytvořenou funkcí *OLEDrgb_backround* z knihovny *pictures.h*. Dalším krokem se přiřadí počáteční hodnota pro vykreslování SpO2 křivky ve spodní části displeje, pak program přechází do části samotného vykreslování křivky, které se provádí pro vertikální osu čtením z pole hodnot a horizontální argument souřadnice se inkrementuje o jedničku při každém průchodu cyklem. Během každého kroku vykreslování se mění hodnota SpO2 a bpm podle vertikální hodnoty souřadnice křivky a tlukot srdce, který se provádí vykreslováním jednotlivých funkcí *OLEDrgb_lilH, OLEDrgb_medH* a *OLEDrgb_bigH* probíhá ve vlastním cyklu, ve kterém se při každém průchodu inkrementuje hodnota proměnné a ta rozhoduje o funkci, která se zavolá. Během každého průchodu cyklem se zkontroluje, zdali se nezměnila hodnota spínače, nebo nebylo stlačeno tlačítko. Ve chvíli, kdy vykreslovaná křivka dojde na konec displeje, tak se vymaže a cyklus se opakuje.



Obrázek 25. Blokové schéma animace zařízení pro pletysmografii

4.4. Knihovna funkcí PmodOLEDrgb

Knihovna PmodOLEDrgb je napsaná v jazyce C, obsahuje funkce pro ovládání barevného OLED displeje.

Definice jednotlivých funkcí mají strukturu:

typ a jméno funkce (deklarace parametrů)

{

deklarace lokálních proměnných funkce

příkazy

}

Knihovna obsahuje 36 funkcí, sloučeny ve skupinách dle jejich významu. Jedná se o skupiny funkcí pro řízení displeje, pro zápis příkazů a dat, ovládání napětí na určitých pinech, grafické funkce a funkce pro zobrazení textu, znaků a definování nových znaků a tabulek písem.

V následujícím textu jsou důležité funkce rozepsány podrobněji i s vysvětlujícími příklady.

4.4.1. Řídící funkce

Tabulka 5. Knihovna řídících funkcí

Funkce		Datový typ	Popis
Spuštění displeje	`OLEDrgb_Begin`	`void`	Inicializuje řadič displeje, zapne jej a nastaví výchozí podmínky.
	`OLEDrgb_HostInit`		Provede inicializaci FPGA (hostitelské desky)
	`OLEDrgb_DevInit`		Projde inicializačními kroky a zapne displej
	`OLEDrgb_SPIInit`	`int`	Nakonfiguruje instanci XSpi pro použití s PmodOLEDrgb
Vypnutí displeje	`OLEDrgb_end`	`void`	Vypíná zařízení, nechává plovoucí piny a zastaví ovladač SPI
	`OLEDrgb_HostTerm`		Zastaví SPI a zanechá plovoucí piny
	`OLEDrgb_DevTerm`		Vypne displeje
	`OLEDrgb_EnablePmod`		Zapíná nebo vypíná displej podle zadaného parametru.
Zápisu dat	`OLEDrgb_WriteSPICommand`		Zapíše byte přes SPI
	`OLEDrgb_WriteSPI`		Zapisuje série příkazů následovaných daty přes SPI
Podsvícení	`OLEDrgb_EnableBackLight`	`void`	Zapíná nebo vypíná podsvícení displeje

FunkceOLEDrgb_HostInit

```
voidOLEDrgb_HostInit(PmodOLEDrgb* InstancePtr)
```

OLEDrgb_HostInit, aby mohla inicializovat hostitelskou desku FPGA, musí zavolat funkci z knihovny *xspi.h*. *XSpi_Start*, aby spustila přenos mezi deskou a displejem (tato funkce je dále rozebrána v kapitole 4.5). *XSpi_IntrGlobalDisable*, toto makro zakáže globální přerušení zařízení tak, aby pracovalo v režimu dotazování (polled mode).

FunkceOLEDrgb_DevInit

```
voidOLEDrgb_DevInit(PmodOLEDrgb* InstancePtr)
```

Inicializuje řadič displeje OLEDrgb a zapne displej, aby toho docílila, musí zavolat funkci *Xil_Out32*z knihovny *xil_io.h*, abynastavila Pmod na hodnotu HIGH, provede se obnovení výchozího stavu a projde kroky inicializace (Displej se vypne, nastaví se přemapování a formát dat, počáteční řádek zobrazování, nastaví hodnoty napětí a frekvenci hodinového signálu, kontrast barev, vypne posunování kontextu na displeji, …) , vymaže obsah displeje (*OLEDrgb_Clear*, blíže vysvětlena v kapitole 4.4.3, přivede napájecí napětíV_{cc} a počká, až se napětí ustálí na hodnotě.

FunkceOLEDrgb_SPIInit

```
intOLEDrgb_SPIInit(XSpi *SpiInstancePtr)
```

Zavolají se funkce z knihovny *xspi.h*, nejprve funkce konfigurující SPI *XSpi_CfgInitialize* a pak se zavolá funkce nastavení možností *XSpi_SetOptions* a výběru zařízení v režimu slave *XSpi_SetSlaveSelect* (těmito funkcemi se blíže zabývá kapitola 4.5). Po provedení funkce je zařízení připraveno na komunikaci přes SPI.

Funkce OLEDrgb_Begin

```
voidOLEDrgb_begin(PmodOLEDrgb* InstancePtr, u32 GPIO_Address, u32 SPI_Address)
```

Realizuje inicializaci řídícího obvodu a zapnutí displeje. Pro dosážení tohoto stavu se nejprve přiřadí adresa zařízení pro komunikaci přes SPI, dále se vymezí bitová velikost jednotlivých znaků. Funkce určí maximální hodnotu vertikálního argumentu souřadnice (řádku) a maximální hodnotu argumentu horizontální souřadnice (sloupce), tato část také definuje, kolik znaků se vejde na displej. Zavolají se funkce pro nastavení fontů, barvy pozadí a barvy vykreslovaných znaků, blíže vysvětlené v kapitole 4.4.3. Nakonec se zavolají funkce *OLEDrgb_SPIInit, OLEDrgb_HostInit, OLEDrgb_DevInit*, které jsou výše v této kapitole popsány.

Funkce OLEDrgb_HostTerm

```
voidOLEDrgb_HostTerm(PmodOLEDrgb* InstancePtr)
```

Zavolá funkci *XSpi_Stop* z knihovny *xspi.h* (funkce je blíže popsána v kapitole 4.5), která ukončí komunikaci přes SPI rozhraní a funkce *Xil_Out32* z knihovny *xil_io.h*, nastaví hodnotu signálových pinů a pinů regulujících výkon na vstupy.

Funkce OLEDrgb_End

```
voidOLEDrgb_end(PmodOLEDrgb* InstancePtr)
```

Ve funkci *OLEDrgb_End*se volají dvě výše podrobněji vysvětlené funkce *OLEDrgb_HostTerm* a *OLEDrgb_DevTerm*. Smyslem funkce je ukončení programu.

Funkce OLEDrgb_WriteSPI

```
voidOLEDrgb_WriteSPI(PmodOLEDrgb* InstancePtr, u8 *pCmd, int nCmd, u8 *pData,int
```

S využitím funkcí z knihovny *xspi.h*, zapíše přes SPI rozhraní sérii příkazů následovaných daty. *XSpi_Transfer*, která je blíže popsána v kapitole 4.5. Pokud jsou nějaká data k přenosu, zapíše se jejich hodnota funkcí *Xil_Out32* na potřebnou adresu.

Funkce OLEDrgb_WriteSPICommand

```
voidOLEDrgb_WriteSPICommand(PmodOLEDrgb* InstancePtr, u8 cmd)
```

Jedinou volanou funkcí *XSpi_Transfer* (z dále blíže popsané knihovny *xspi.h*) se provede přenos příkazu o velikosti 1 bajtu přes SPI do odesílajícího bufferu. Parametr cmd je příkaz k odeslání.

FunkceOLEDrgb_EnableBackLight

```
voidOLEDrgb_EnableBackLight(PmodOLEDrgb* InstancePtr, u8 fEnable)
```

Zapíná nebo vypíná se podsvícení tím, že se nastaví napětí V_{CCRef} na pinu na hodnotu HIGH nebo LOW a vstoupí buď do režimu Display on, nebo off. Všechna nastavení a vizuální kontext dipsleje jsou zachovány, když je podsvícení opět povoleno po jeho vypnutí.

4.4.2. **Pro zobrazení znaků a textu**

Tabulka 6. Knihovna funkcí pro zobrazení znaků a textu

Funkce		**Datový typ**	**Popis**
Nastavení kurzoru	`OLEDrgb_SetCursor`	`void`	Nastaví souřadnice pozice kurzoru na zvolené místo na displeji
	`OLEDrgb_GetCursor`		Zjistí aktuální pozici kurzoru
	`OLEDrgb_AdvanceCursor`		Kurzor se posune o jeden znak po vykreslení, aby se dalším znakem původní nepřepsal.
Nastavení znaku	`OLEDrgb_DefUserChar`	`int`	Zadání definice piktogramu pro znakový kód zadaného uživatele.
	`OLEDrgb_DrawGlyph`	`void`	Vykreslí zadaný znak do vyrovnávací paměti (bufferu) displeje v aktuální pozici kurzoru.
	`OLEDrgb_PutChar`		Vloží zadaný znak na displeji v aktuální pozici kurzoru a posune kurzor.
	`OLEDrgb_PutString`		Napíše zadaný řetězec znaků ukončený nulou na displej a posuňte kurzor.
Nastavení barev	`OLEDrgb_SetFontColor`		Nastaví barvu, která se použije jako barva písma při tisku znaků.
	`OLEDrgb_SetFontBkColor`		Nastaví barvu, která bude použita jako barva pozadí písma při tisku znaků.
Nastavení tabulky fontů	`OLEDrgb_SetCurrentFont Table`		Definuje novou tabulku písem.
	`OLEDrgb_SetCurrent UserFontTable`		Definuje novou tabulku písem uživatelů.
Převádění hodnot HSV	`OLEDrgb_BuildHSV`	`uint 16_t`	Převede hodnotu HSV na spektrum barvy 565 RGB, kterou používá displej OLEDrgb. Návratová hodnota reprezentuje hodnotu RGB vstupní barvy v 16 bitovém barevném formátu. Parametr hue znamená odstín barvy, sat je saturace barvy a val je RGB hodnota barvy.
Převádění hodnot RGB	`OLEDrgb_BuildRGB`		Převede jednotlivé 8 bitové hodnoty barev na 16 bitové hodnoty spektra RGB, kterou používá PmodOLEDrgb.
Vyjmutí barvy	`OLEDrgb_ExtractXFromRGB`		Odstraní určitou část barevného spektra

Funkce OLEDrgb_AdvanceCursor

```
voidOLEDrgb_AdvanceCursor(PmodOLEDrgb* InstancePtr)
```

Přičte 1 k aktuálnímu horizontálnímu argumentu souřadnice pozice kurzoru. Poté se zjišťuje, zdali je nynější pozice kurzoru na displeji, tzn. hodnoty jeho souřadnic budou menší nebo rovny maximálně dosažitelné hodnotě *(xchOledrgbMax, ychOledrgbMax)* argumentu souřadnice pozice na displeji. Jestliže je horizontální souřadnice v tomto rozmezí, vertikální se posune o 1. Je-li vertikální souřadnice v rozmezí, přičte se k horizontální souřadnici 1. Čímž se kurzor znaku posune o jedno znakové pole.

Funkce OLEDrgb_BuildRGB

```
u16OLEDrgb_BuildRGB(u8 R, u8 G, u8 B)
```

Převede samostatné 8bitovéhodnoty RGB na 16 bitovéhodnotu (565) RGB, kterou používá OLEDrgb displej.

Příklad:

Převádí-li se kompletně zelený kanál pak, na všech pozicích bude 1.

Tabulka 7. Plně zelený 8 bitový kanál

1	1	1	1	1	1	1	1

Funkce OLEDrgb_BuildRGB definuje, že zelená je v rozmezí ((G>>2)<<5), takže seo tyto pozice (tzn. bitovým posunem) posunou nejprve o 2 vpravo a o 5 vlevo v 16 bitové spektru. Červený by se posouval (R>>3)<<11 a modrý (B>>3)

Tabulka 8. Plně zelený 16 bitový kanál

0	0	1	1	1	1	1	1	→	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0

Funkce OLEDrgb_DrawGlyph

```
voidOLEDrgb_DrawGlyph(PmodOLEDrgb* InstancePtr, char ch)
```

Je-li hodnota kódu znaku větší než nula, ale menší než maximální definovaná velikost znaku, provede se vymaskování, čímž se zjistí, jaké hodnoty jsou na jednotlivých pozicích bitmapy, podle těchto výsledků se poté řeší barvy znaku a pozadí. Je-li v bodě hodnota 1, volí se barva písma. V opačném případě s hodnotou 0 se volí barva pozadí znaku. Čímž se vykreslí zadaný znak do vyrovnávací paměti (bufferu) displeje v aktuálním umístění kurzorů znaků. To neovlivní aktuální pozici kurzoru znaků ani aktuální pozici výkresu ve vyrovnávací paměti displeje, to znamená, že pokud se bude vykreslovat další znak, původní se jím překreslí.

Funkce OLEDrgb_DefUserChar

```
int OLEDrgb_DefUserChar(PmodOLEDrgb* InstancePtr, char ch, uint8_t * pbDef)
```

Definice znaku spočívá ve vytvoření bitmapové grafiky (bitové mapy) jednotlivých použitých znaků/písmen v tomto případě srdce a EKG křivka. V poli se nachází 8 x 8 bitů popisujících bitovou mapu znaku. Konkrétně u znaku srdce se jedná o data 0x00, 0x0C, 0x12, 0x22, 0x4C, 0x22, 0x12, 0x0C. Jednotlivé hodnoty (bajty) vyjadřují vždy 1 sloupec ve výsledné bitové mapě, s tím že první bajt je první sloupec a poslední bajt popisuje zase poslední sloupec. Převedením uvedených hexadecimálních čísel do binární podoby získáme bitovou mapu znázorněnou na Obrázek 26.

	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7	
0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	1	0		0	0	0	0	0	0	0	0	**0**
1	0	0	1	1	0	1	1	0		0	0	0	0	0	0	1	0		0	0	0	0	0	0	0	0	**1**
2	0	1	0	0	1	0	0	1		0	0	0	0	0	0	1	0		0	0	0	0	0	0	0	0	**2**
3	0	1	0	0	1	0	0	1		0	0	1	0	0	0	1	0		0	0	0	1	0	0	0	0	**3**
4	0	0	1	0	0	0	1	0		0	1	1	1	0	1	0	1		0	0	1	0	1	0	0	0	**4**
5	0	0	0	1	0	1	0	0		1	1	0	1	0	1	0	1		0	1	1	0	1	1	1	1	**5**
6	0	0	0	0	1	0	0	0		0	0	0	0	1	0	0	0		1	0	0	0	0	0	0	0	**6**
7	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0		1	0	0	0	0	0	0	0	**7**
	0x00	**0x0C**	**0x12**	**0x22**	**0x4C**	**0x22**	**0x12**	**0x0C**		**0x20**	**0x30**	**0x18**	**0x30**	**0x40**	**0x30**	**0x0F**	**0x30**		**0xC0**	**0x20**	**0x30**	**0x08**	**0x30**	**0x20**	**0x20**	**0x20**	

Obrázek 26. Bitová mapa srdce a ekg křivky (3 znaky)

Funkce OLEDrgb_PutChar

```
void OLEDrgb_PutChar(PmodOLEDrgb* InstancePtr, char ch)
```

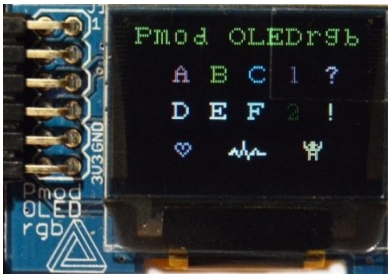
Využitím funkce *OLEDrgb_DefUserChar*, byly znaky definovány a funkce *OLEDrgb_SetFontCol*or nastaví barvu znaku podle nastavení parametrů, jinak je znak defaultně zelený. *OLEDrgb_PutChar* vykreslí znak na displej *(OLEDrgb_DrawGlyph)* na aktuální pozici kurzoru a posune kurzor za znak *(OLEDrgb_AdvanceCursor).*

Obrázek 27. Vykreslené znaky na displeji

Funkce OLEDrgb_PutString

```
void OLEDrgb_PutString(PmodOLEDrgb* InstancePtr, char * sz)
```

Zapíše specifikovaný řetězec znaků pomocí *OLEDrgb_DrawGlyph* na displej a *OLEDrgb_Advance Cursor* posune kurzor za vykreslený obsah, takže se při vykreslení dalšího znaku, původní znak nepřepíše. Znaky jsou definovány v knihovně *ChrFont0.h*.

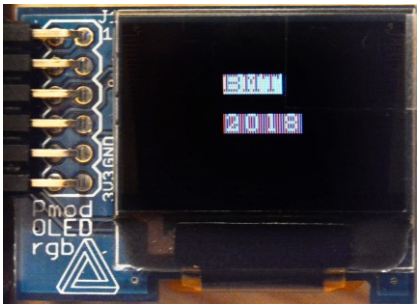


Obrázek 28. Vizualizace funkcí OLEDrgb PutString, SetFontColour a PutChar

Funkce OLEDrgb_SetBkColor

```
void OLEDrgb_SetFontBkColor(PmodOLEDrgb* InstancePtr, uint16_t fontBkColor)
```

Nastavuje barvu, která se použije jako barva pozadí písma při tisku znaků. Blíže popsáno ve výše popsané funkci *OLEDrgb_DrawGlyph*.



Obrázek 29. Vizualizace funkce OLEDrgb_SetBkColor

4.4.3. **Grafické funkce**

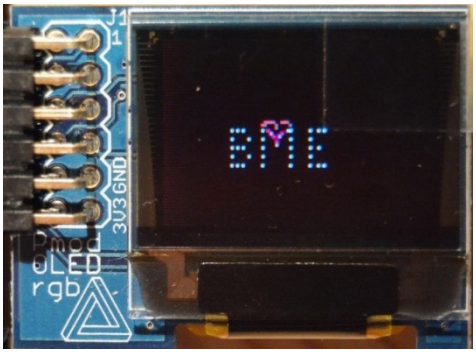
Tabulka 9. Knihovna grafických funkcí

Funkce		Datový typ	Popis
Vykreslování	`OLEDrgb_DrawPixel`	`void`	Nakreslí bod na specifikované pozici ve zvolené barvě
	`OLEDrgb_DrawLine`		Nakreslí barevnou čáru ze zadané výchozí pozice do zadané koncové polohy
	`OLEDrgb_DrawRectangle`		Nakreslí barevný obdélník na uživatelem specifikované ploše
	`OLEDrgb_DrawBitmap`		Vykreslí rastrový obrázek do uživatelem zvolené oblasti na displeji
Vymazání displeje	`OLEDrgb_Clear`		Displej a vyrovnávací paměť displeje se vymažou a následně se zobrazení aktualizuje. To znamená, že obsah dat v RAM displeje bude nastaven na nulu.
Nastavení posouvání	`OLEDrgb_SetScrolling`		Konfiguruje posouvání podle zadaných parametrů.
	`OLEDrgb_EnableScrolling`		Aktivuje nebo zakazuje možnost posouvání.
Kopírování	`OLEDrgb_Copy`		Zkopíruje obsah vybrané oblasti do nového umístění
	`OLEDrgb_Dim`		Ztmaví barvu specifikované oblasti

Funkce OLEDrgb_DrawPixel

```
void OLEDrgb_DrawPixel(PmodOLEDrgb* InstancePtr, uint8_t c, uint8_t r, uint16_t
pixelColor)
```

Funkce vykresluje body na specifikovaných souřadnicích reprezentovaných adresou ve vyrovnávací paměti pole. Souřadnice bodu se zadávají jako c a r argumenty funkce. Argument c může nabývat libovolnou hodnotu mezi 0-95, zatímco r může mít libovolnou hodnotu mezi 0-63. Bod se souřadnicemi (0,0) by se nacházel v levé horní části obrazovky. Pokud je souřadnice mimo rozsah, program pokračuje ve vykreslování na dalším řádku.

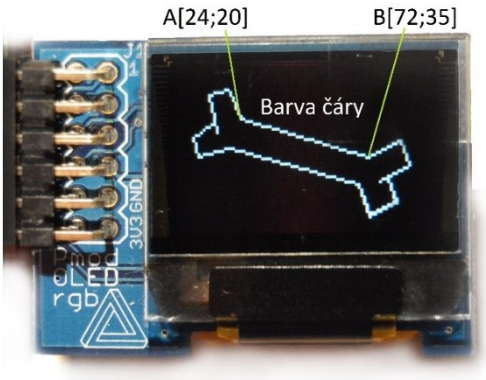


Obrázek 30. Vizualizace funkce OLEDrgb_DrawPixel

Funkce OLEDrgb_DrawLine

```
void OLEDrgb_DrawLine(PmodOLEDrgb* InstancePtr, uint8_t c1, uint8_t r1, uint8_t
                      c2, uint8_t r2, uint16_t lineColor)
```

Funkce vykresluje čáru o specifikovaných souřadnicích počátečního a koncového bodu reprezentovaných adresou ve vyrovnávací paměti pole. Souřadnice počátečního bodu se zadávají jako c1 a r1 argumenty funkce a argumenty souřadnice koncového bodu jsou c2 a r2. Argument c1 a c2 může nabývat libovolnou hodnotu mezi 0-95 (včetně), zatímco r1 a r2 mohou nabývat jakoukoli hodnotu v rozmezí od 0 do 63. Bod se souřadnicemi (0,0) by se nacházel v levé horní části obrazovky. Pokud je souřadnice mimo rozsah, program pokračuje ve vykreslování na dalším řádku. V demonstračním příkladu byla vykreslována bílá čára ze zadané výchozí pozice, což byl bod s hodnotou argumentů c1 = 24 (sloupec) a r1 = 20 (řádek) do zadané koncové polohy, jejíchž hodnoty argumentů jsou c2 = 35 a r2 = 75. Barva čáry je bíla, spektrum této barvy bylo vytvořeno s pomocí funkce z kapitoly 4.4.2 a to *OLEDrgb_Build* s hodnotou argumentů (255, 255, 255).

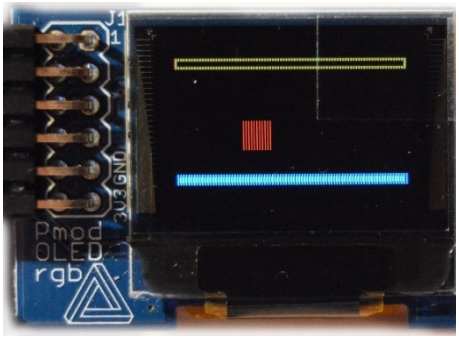


Obrázek 31. Vizualizace funkce OLEDrgb_DrawLine

Funkce OLEDrgb_DrawRectangle

```
void OLEDrgb_DrawRectangle(PmodOLEDrgb* InstancePtr, uint8_t c1, uint8_t r1,
uint8_t c2, uint8_t r2, uint16_t lineColor, bool bFill, uint16_t fillColor)
```

Nakreslí obdélník ze zadané výchozí polohy do zadané koncové polohy pomocí specifikované barvy čáry. Podle parametru bFill je obdélník vyplněn a odstínem barvy podle parametru fillColor. Provede se výběr barvy ohraničení (čar) a výplně, nastaví se souřadnice výchozího a koncového bodu, vykreslí se obdélník a vyplní zvolenou barvou, je-li hodnota „true“ v parametru bFill.

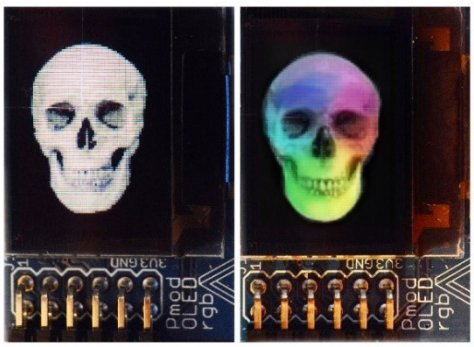


Obrázek 32. Vizualizace funkce OLEDrgb_DrawRectangle

Funkce OLEDrgb_DrawBitmap

```
void OLEDrgb_DrawBitmap(PmodOLEDrgb* InstancePtr, uint8_t c1, uint8_t r1, uint8_t
                        c2, uint8_t r2, uint8_t *pBmp)
```

V bitmapové (rastrové) grafice je celý obrázek popsán pomocí jednotlivých barevných bodů (pixelů). Body jsou uspořádány do mřížky (podobně jako na Obrázek 26), kde každý bod má určenou svou přesnou polohu a barvu v barevném modelu (RGB). Obrázek se vykreslí po zavolání této funkce, jejíž argumenty jsou stejné jako u předchozí funkce počátečního a koncového bodu obdélníku, jen s tím rozdílem, že argument pBmp označuje pointer na 16 bitovou výše zmíněnou mřížku.

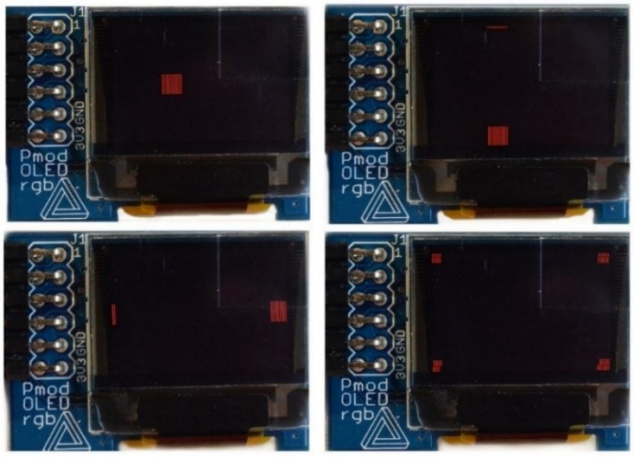


Obrázek 33. Vizualizace funkce OLEDrgb_DrawBitmap

Funkce OLEDrgb_SetScrolling

```
void OLEDrgb_SetScrolling(PmodOLEDrgb* InstancePtr, u8 scrollH, u8 scrollV,
  u8 rowAddr, u8 rowNum, u8 timeInterval)
```

V průběhu funkce se konfiguruje horizontální *(scrollH),* vertikální *(scrollV)* případně diagonální posouvání, adresa počátečního řádku *(rowAddr),* počet řádků *(rowNum),* o které se bude rolovat, časový interval *(timeInterval)* mezi jednotlivými posuny. Posun po diagonále se provádí kombinací parametrů *scrollH* a *scrollV*, mají-li stejnou hodnotu různou od nuly.



Obrázek 34. Vizualizace funkce OLEDrgb_SetScrolling

Původní pozice (vlevo nahoře), Vertikální posun (vpravo nahoře), Horizontální posun (vlevo dole) a Diagonální posun (vpravo dole).

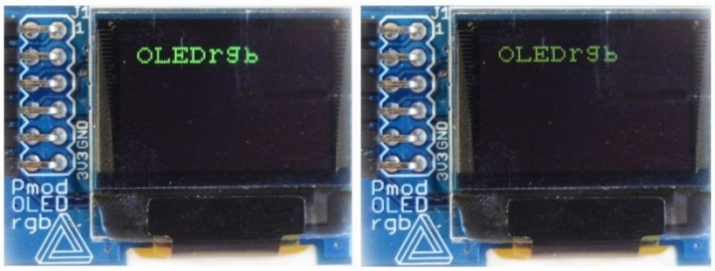
Funkce OLEDrgb_Dim

```
void OLEDrgb_Dim(PmodOLEDrgb* InstancePtr, u8 c1, u8 r1, u8 c2, u8 r2)
```

Parametry jsou stejné jako u funkce*O° LEDrgb_DrawRectangle*. Po provedení tohoto příkazu bude vybrané okno tmavší, podle Tabulka 10, dle principu přiblíženého v kapitole 2.6.3.

Tabulka 10. Nová stupnice šedi pro ztmavenou oblast

Originální stupnice šedi	Nová stupnice šedi pro ztmavenou oblast
GS0 ~ GS15	Beze změny
GS16 ~ GS19	GS4
GS20 ~ GS23	GS5
:	:
GS60 ~ GS63	GS15

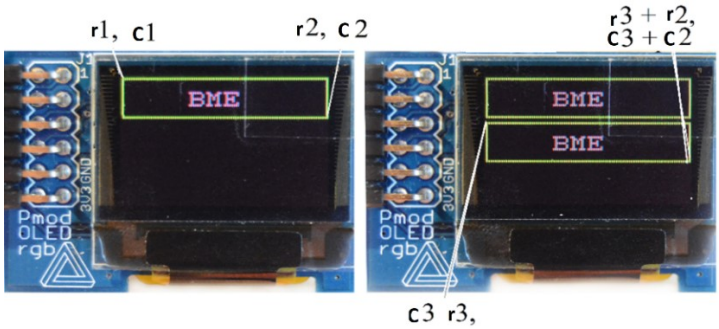


Obrázek 35. Vizualizace funkce OLEDrgb_Dim (Původní (vpravo), Ztmavená (vlevo))

Funkce OLEDrgb_Copy

```
void OLEDrgb_Copy(PmodOLEDrgb*InstancePtr,u8 c1,u8 r1, u8 c2, u8 r2, u8 c3, u8 r3)
```

Zvolené parametry přiřadí hodnoty souřadnicím počátečního (r1 – řádek, c1 – sloupec) a koncového bodu (r2, c2), které se překopíruje do nového umístění určeného souřadnicemi bodu (r3, c3). Pokud jsou nové souřadnice v oblasti původního obrazu, nový obraz ten původní překryje. Tímto postupem se zkopírujte obsah oblasti vymezené obdélníkem do nového umístění. V příkladu níže je oblast, která má být zkopírována pro názornější vizualizaci zviditelněna obdélníkem vykresleným funkcí *OLEDrgb_DrawRectangle*, jinak v této funkci kopírovaná oblast není nijak zvýrazněna.



Obrázek 36. Vizualizace funkce OLEDrgb_Copy (Původní (vpravo), Stav po zkopírování (vlevo))

4.5. Knihovna funkcí SPI

Knihovna xspi obsahuje funkce potřebné ke komunikaci přes SPI. Uvnitř knihovny se nachází 12 funkcí zprostředkovávajích přenos příkazů a dat po sběrnici SPI. Bližší popis SPI se nachází v kapitole 3.5.

Tabulka 11. Knihovna funkcí SPI

Funkce		Datový typ	Popis
Inicializace SPI	`XSpi_Initialize`	`int`	Inicializuje SPI rozhraní
	`XSpi_CfgInitialize`		Inicializuje specifickou instanci Xspi
Ovládající přenos	`XSpi_Start`		Spouští SPI komunikaci
	`XSpi_Stop`		Zastavuje SPI komunikaci
	`XSpi_Transfer`	`int`	Slouží pro přenos příkazů a dat
	`XSpi_Reset`	`void`	Znovu nastavení výchozích podmínek přenosu, konfigurace po resetu je stejná jako po jeho inicializaci a veškerý přenos dat je ukončen
	`XSpi_Abort`		Přerušení přenosu
Zařízení v režimu slave	`XSpi_SetSlaveSelect`		Zvolí nebo zruší výběr podřízeného zařízení
	`XSpi_GetSlaveSelect`	`u32`	Získá bitovou masku aktuálně vybraného podřízeného zařízení.
Ovládající handler	`XSpi_SetStatusHandler`	`void`	Nastaví ukazatele na funkci, jenž je zodpovědná za zpracování vyvolaných stavů v aplikaci
	`XSpi_InterruptHandler`		Zpracování chyb pří přenosu příkazů a dat
	`StubStatusHandler`	`static void`	Zástupný ukazatel na funkci v případě, že vyšší vrstva SW jej nenastaví

Funkce XSpi_CfgInitialize

```
int XSpi_CfgInitialize(XSpi *InstancePtr, XSpi_Config *Config,
                       u32 EffectiveAddr)
```

Inicializuje konkrétní XSpi instanci tak, aby byl ovladač připraven k použití. Pokud je zařízení již inicializováno, nepovolí opakovanou inicializaci a vrátí stav indikující jeho spuštění, což brání neúmyslnému inicializování. Nastaví výchozí hodnoty. Resetuje SPI zařízení, aby se dostalo do původního stavu. Očekává se, že konfigurace zařízení proběhne po dokončení inicializace, ale před spuštěním zařízení. Funkce inicializuje instanci *InstancePtr* pro konkrétní zařízení určené obsahem konfigurace. Parametr *Config* je odkaz na strukturu obsahující informace o konkrétním SPI zařízení. *EffectiveAddr* je adresa zařízení v adresním prostoru virtuální paměti.

Funkce XSpi_Start

```
int XSpi_Start(XSpi *InstancePtr)
```

Funkce spouští SPI zařízení. Pokud je řadič SPI používán v režimu přerušení (interrupt mode), je na uživateli, aby před voláním této funkce zajistil obsluhu příslušného přerušení. Pokud je ovladač SPI používán v režimu dotazování (polled mode), musí uživatel po ukončení této funkce zakázat globální přerušení. Je-li zařízení nakonfigurováno s FIFO, jsou FIFO v tomto okamžiku vynulovány. Návratovými hodnotami jsou buď XST_SUCCESS, což znamená, že zařízení bylo úspěšně zapnuto, nebo XST_DEVICE_IS_STARTED, což znamená, že zařízení již bylo zapnuto při volání funkce.

Funkce XSpi_Stop

```
int XSpi_Stop(XSpi *InstancePtr)
```

Funkce vypíná SPI zařízení vypnutím přerušení a vypnutím samotného zařízení. Přerušení jsou zakázány pouze v rámci samotného zařízení. V režimu přerušení, je-li zařízení v procesu přenosu dat na SPI sběrnici, pak funkce vrátí, že je zařízení zaneprázdněno. Uživatel bude upozorněn prostřednictvím obslužného stavu, jakmile bude přenos dokončen a v tomto okamžiku se může pokusit zastavit zařízení.

V režimu master není dovoleno zastavit zařízení během přenosu dat, neboť to může vést k neznámým stavům na straně slave zařízení.

V režimu slave není dovoleno zastavit zařízení během přenosu dat, protože master ještě neukončil přenos.

Funkce XSpi_Reset

```
void XSpi_Reset(XSpi *InstancePtr)
```

Resetuje SPI zařízení zápisem do Software reset registru. Reset může být volán pouze po dokončení inicializace daného SPI zařízení ovladače. Konfigurace zařízení po resetování je stejná jako jeho konfigurace po inicializaci. Jedná se o tvrdý reset zařízení – jakýkoli přenos dat, který probíhá, bude přerušen. Software vyšší vrstvy je zodpovědný za opětovnou konfiguraci (v případě potřeby) a restartování SPI zařízení.

Funkce XSpi_SetSlaveSelect

```
int XSpi_SetSlaveSelect(XSpi *InstancePtr, u32 SlaveMask)
```

Zvolí nebo zruší výběr podřízeného zařízení (slave), s nímž komunikuje řídící zařízení (master). Každé podřízené zařízení, které je možné vybrat, je reprezentováno v registru pro výběr podřízené jednotky jedním bitem. Argument předán této funkci je bitová maska se zvolením 1 v bitové pozici pro vybrání slave. Může být vybrán pouze jeden slave v jeden moment. Uživatel nemá povoleno zrušit výběr slave během probíhajícího přenosu. Pokud neprobíhá žádný přenos, může uživatel vybrat nové slave zařízení, které zruší výběr aktuálního slave. Pokud je požadováno zrušení výběru aktuálního slave zařízení, může být jako hodnota argumentu pro danou funkci předána 0. Tato funkce pouze označuje, který slave bude vybrán při zahájení přenosu. Podřízené zařízení není vybráno, pokud je SPI nečinné.

Funkce XSpi_GetSlaveSelect

```
u32 XSpi_GetSlaveSelect(XSpi *InstancePtr)
```

Získá aktuální bitovou masku sběrnice vybraného slave zařízení pro SPI zařízení. Funkce zjišťuje pouze stav nastavený funkcí *XSpi_SetSlaveSelect,* nezjišťuje stav samotného registru jádra. Návratová hodnota je 32bitová hodnota maskovaná s 1 na bitové pozici aktuálně vybraného slave zařízení. Hodnota může být nula, pokud nejsou vybrány žádné zařízení typu slave.

Funkce XSpi_Transfer

```
int XSpi_Transfer(XSpi *InstancePtr, u8 *SendBufPtr, u8 *RecvBufPtr, unsignedint
ByteCount)
```

Funkce přenáší zadaná data po sběrnici SPI. Pokud je zařízení nakonfigurováno jako master (řídící zařízení), pak funkce iniciuje komunikaci a odesílá nebo přijímá data do nebo z vybraného slave (podřízeného zařízení). Pokud je zařízení nakonfigurováno jako slave, pak funkce připraví data, která mají být odeslána nebo přijata, jakmile jsou požadována mastrem (řídícím zařízením).

Ovladač pracuje v režimu dotazování a přerušení.

1. V režimu přerušení je funkce neblokující – přenos je iniciován touto funkcí a dokončen obsluhou přerušení.
2. V režimu dotazování je tato funkce blokující a program opustí tuto funkci až po přenesení všech požadovaných dat.

Je možné mít dva různé buffery (vyrovnávací paměti) pro odesílání a příjem nebo jeden buffer pro obojí, nebo jen jeden pro odesílání. Buffer pro příjem musí být alespoň tak velký jako ten pro odesílání, aby se zabránilo nevyžádanému zápisu do paměti. To znamená, že počet bajtů předaných v argumentu musí být menší, než ve druhém bufferu (pokud se liší ve velikosti). Je-li zařízení v režimu master musí být před touto funkcí volána funkce *XSpi_SetSlaveSelect*.

V režimu přerušení, může přenášet pouze omezený počet bajtů v čase. Pokud není použité žádné FIFO, přenáší najednou pouze jeden bajt. Pokud je použito FIFO, může přenést najednou více bajtů, až do velikosti daného FIFO. V režimu přerušení spustí volání této funkce pouze přenos. Následný přenos dat se provádí rutinou obsluhy přerušení až do doby, kdy je celý buffer přenesen.

V režimu dotazování je tato funkce blokující a řízení ukončí funkci až po přenesení všech požadovaných dat.

Návratové hodnoty jsou XST_SUCCESS, jestliže byl přenos úspěšný, XST_DEVICE_IS_STOPPED, pokud bylo zařízení zastaveno během přenosu, XST_DEVICE_BUSY, pokud je zařízení zaneprázdněné a XST_SPI_NO_SLAVE, značí, že nebylo vybráno a nakonfigurováno slave zařízení.

Funkce XSpi_Abort

```
void XSpi_Abort(XSpi *InstancePtr)
```

Přeruší probíhající přenos nastavením stop bitu v řídícím registru a resetováním FIFO (datová struktura first in, first out). Počítač bajtů je vymazán a příznak zaneprázdnění je nastaven na hodnotu false. Tato funkce provádí čtení, změnu nebo zápis do řídicího registru.

Funkce XSpi_SetStatusHandler

```
void XSpi_SetStatusHandler(XSpi *InstancePtr, void *CallBackRef,
                           XSpi_StatusHandler FuncPtr)
```

Nastaví ukazatel na funkci, jenž je zodpovědná za zpracování vyvolaných stavů v procesu.

XST_SPI_MODE_FAULT – Objevila se chyba zvoleného módu (master/slave) – jiný master se pokoušel vybrat toto zařízení jako slave i přesto, že je toto zařízení nakonfigurováno jako master. Jakýkoli probíhající přenos je přerušen.

XST_SPI_TRANSFER_DONE – Požadovaný přenos dat je hotov.

XST_SPI_TRANSMIT_UNDERRUN – V režimu slave, master vyžadoval data, ale žádná nebyla k dispozici. To obvykle znamená, že podřízená (slave) aplikace nevydala požadavek na přenos dostatečně rychle, nebo že procesor nebo řadič nestihl dostatečně rychle zaplnit přenosový registr.

XST_SPI_RECEIVE_OVERRUN – SPI zařízení ztratilo data, tzn. data byla přijata, ale registr přijatých dat byl již plný. To znamená, že zařízení přijímá data rychleji, než procesor je schopen data zpracovávat.

XST_SPI_SLAVE_MODE_FAULT – Ke komunikaci bylo vybráno slave zařízení, které je však deaktivováno. Obvykle to znamená, že master již přenáší data, ta jsou ale ztracena. Je potřeba, aby si slave zařízení vyžádalo přenos.

Funkce XSpi_ InterruptHandler

```
void XSpi_InterruptHandler(void *InstancePtr)
```

Zpracování chyb pří přenosu příkazů a dat. Funkce musí být spojena se zdrojem přerušení. Neukládá a neobnovuje kontext procesoru, takže toto musí být zařízeno uživatelem. Zpracovatelná přerušení jsou:

Chyba zvoleného režimu – Přerušení se generuje, je-li toto zařízení vybráno jako slave, když je nakonfigurováno jako master. Řadič přeruší veškerý probíhající datový přenos, resetováním jeho ukazatelů ve vyrovnávací paměti. Software vyšší vrstvy je informován o chybě.

Chyba nastavení režimu slave – Přerušení se vygeneruje, pokud je zařízení vybráno jako slave v době, kdy není aktivní. Žádné akce nejsou prováděny, pouze je o chybě informována vyšší softwarová vrstva.

Přerušení odvislé od registru přenosu dat:

- *Prázdný registr:* Přerušení se generuje, je-li registr přenosu dat prázdný. Ovladač využívá tohoto přerušení během přenosu, aby nepřetržitě posílal nebo přijímal data, až do chvíle, kdy nebudou žádná další data k odeslání, či příjmu.
- *Nedostatečně zaplněný přenosový registr:* Přerušení se generuje, když SPI zařízení je nakonfigurováno jako slave a pokusí se číst prázdný buffer. Prázdný buffer DTR/FIFO obvykle znamená, že software neposkytuje data včas. Žádné akce nejsou prováděny, pouze je o chybě informována vyšší softwarová vrstva.
- *Přeplnění bufferu:* Přerušení se generuje, jakmile se SPI zařízení pokusí zapsat přijatá data do již plného bufferu. Zaplněný buffer obvykle znamená, že nebyl včas vyprázdněn. Žádné další akce nejsou prováděny, pouze je o chybě informována vyšší softwarová vrstva.
- *Přeplnění bufferu:* Přerušení se generuje, jakmile se SPI zařízení pokusí zapsat přijatá data do již plného bufferu. Zaplněný buffer obvykle znamená, že nebyl včas vyprázdněn. Žádné další akce nejsou prováděny, pouze je o chybě informována vyšší softwarová vrstva.

4.6. Knihovna funkcí GPIO

Tabulka 12. Knihovna funkcí GPIO

Funkce		Datový typ	Popis
Inicializace GPIO	`XGpio_Initialize`	`int`	Inicializuje instanci Xgpio na základě dodaného parametru DevideID
	`XGpio_CfgInitialize`		Inicializuje instanci Xgpio na základě dodaných konfiguračních dat
Směr dat	`XGpio_SetDataDirection`	`void`	Nastaví směr datového kanálu GPIO modulu
	`XGpio_GetDataDirection`	`u32`	Zjistí směr dat v datovém kanálu GPIO modulu
Ovládající kanály	`XGpio_DiscreteRead`		Přečte aktuální hodnotu na určitém kanálu
	`XGpio_DiscreteWrite`	`void`	Zapíše hodnotu na specifikovaný kanál
	`XGpio_DiscreteSet`		Nastaví výstup specifikovaného kanálu GPIO na logickou 1.
	`XGpio_DiscreteClear`		Nastaví výstup specifikovaného kanálu GPIO na logickou 0.

Funkce XGpio_Initialize

```
int XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId);
```

Inicializuje instanci XGpio na základě identifikátoru DeviceID. DeviceID je jedinečný identifikátor zařízení řízeného touto instancí. Návratová hodnota je XST_SUCCESS, pokud byla inicializace úspěšná nebo XST_DEVICE_NOT_FOUND, pokud nebyla nalezena konfigurační data zařízení pro zařízení s dodaným ID zařízení. Ovladač vyhledá vlastní konfigurační strukturu vytvořenou řetězce nástrojů na základě ID poskytnutého řetězcem nástrojů.

Funkce XGpio_LookupConfig

```
XGpio_Config *XGpio_LookupConfig(u16 DeviceId);
```

Tabulka ConfigTable obsahuje informace o konfiguraci pro každé zařízení v systému. Ukazatel datového typu XGpio_Config odkazuje na konfiguraci zařízení, pokud je nalezen identifikátor DeviceID. Pokud nalezen nebyl, pak bude návratovou hodnotou NULL.

Funkce XGpio_CfgInitialize

```
int XGpio_CfgInitialize(XGpio *InstancePtr, XGpio_Config * Config,
                        u32 EffectiveAddr);
```

Inicializujte instanci XGpio na základě daných konfigurační struktury XGpio. EffectiveAddr je adresa zařízení ve virtuální paměti adresového prostoru. Config je odkaz na strukturu obsahující informace o konfiguraci konkrétního zařízení GPIO. Návratovou hodnotou pokud byla inicializace úspěšná je XST_SUCCESS.

Funkce XGpio_SetDataDirection

```
void XGpio_SetDataDirection(XGpio *InstancePtr, unsigned Channel,
                            u32 DirectionMask);
```

Nastaví směr kanálů GPIO modulu. GPIO obsahuje dva maximálně dva kanály, se kterými je možné pracovat. DirectionMask je bitová maska specifikující, které kanály jsou vstupní a které jsou výstupní. Je-li hodnota bitové masky nastavená na 0 (0x0) jedná se o výstupní kanál a vstupní je pokud jsou bity nastaveny na hodnotu 1 (0xF). Jestliže je tato funkce použita s jiným kanálem než prvním, musí být hardware uzpůsoben pro dva kanály.

Funkce XGpio_GetDataDirection

```
u32 XGpio_GetDataDirection(XGpio *InstancePtr, unsigned Channel);
```

Získá směr všech diskrétních signálů (vstupní, výstupní) pro zadaný kanál GPIO. Vrátí bitovou masku specifikující, které kanály jsou vstupy a které jsou výstupy. Bity nastavené na 1 jsou vstupy a výstupy jsou nastaveny na hodnotu 0.

Funkce XGpio_DiscreteRead

```
u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
```

Přečte stav signálu pro zadaný kanál GPIO. GPIO obsahuje kanály (1 nebo 2), se kterými se dá pracovat. Návratová hodnota je aktuální kopii registru signálů na 1 anebo 2 kanálu.

Funkce XGpio_DiscreteWrite

```
void XGpio_DiscreteWrite(XGpio * InstancePtr, unsigned Channel, u32 Data);
```

Zapisuje data do registru pro zadaný kanál GPIO. Data jsou hodnota, která má být zapsána do registru.

4.7. Komunikační funkce

K samotné komunikaci přes AXI sběrnici jsou k dispozici funkce, které jsou zahrnuty v knihovně *xil.io.h*. Zde funkce *Xil_In*, slouží pro čtení hodnot z adresy, která je jejím parametrem. Návratovou hodnotu funkce tvoří hodnota, která se nachází na této adrese. Funkce *Xil_Out*, má dva parametry a to požadovanou hodnotu a adresu, na kterou ji má zapsat. Uvnitř knihovny se nachází jak vstupní tak výstupní funkce, pro různé bitové šířky.

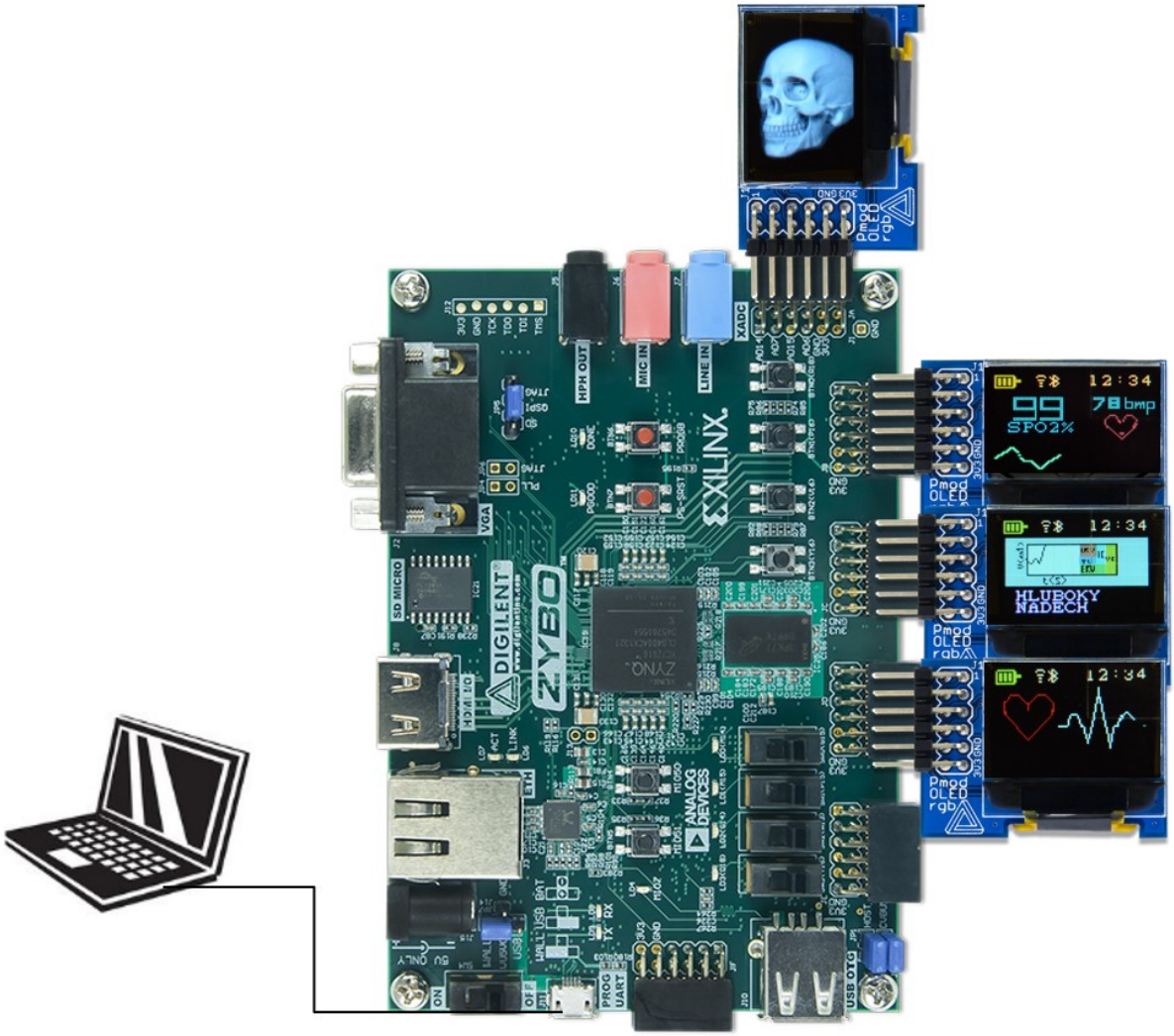
Tabulka 13. Knihovna funkcí IO

Funkce		Datový typ	Popis
Vstup	Xil_In8	u8	Provede čtení ze specifikované adresy 8 bitové oblasti v paměti. Návratovou hodnotou je hodnota odečtená z této adresy.
	Xil_In16	u16	Provádí čtení ze specifikované adresy 16 bitové oblasti v paměti. Návratovou hodnotou je hodnota odečtená z této adresy.
	Xil_In32	u32	Provede čtení ze specifikované adresy 32 bitové oblasti v paměti. Návratovou hodnotou je hodnota odečtená z této adresy.
	Xil_In16BE	u16	Slouží ke čtení hodnot ze zvolené adresy 16 bitové oblasti v paměti a přeskupování 16 bitů oblasti v paměti.
	Xil_In32BE	u32	Slouží ke čtení hodnot z určené adresy 32 bitové oblasti v paměti a provádí přeskupení 32 bitů v paměti.
Výstup	Xil_Out8	void	Provádí zápis specifikované hodnoty na zadanou adresu 8 bitové oblasti v paměti.
	Xil_Out16		Provede zápis specifikované hodnoty na zadanou adresu 16 bitové oblasti v paměti.
	Xil_Out32		Provádí zápis specifikované hodnoty na zadanou adresu 32bitové oblasti v paměti.
	Xil_Out16BE		Slouží k přeskupení bajtů v paměti a zápisu zadaných hodnot na určitou adresu.
	Xil_Out32BE		Slouží k přemisťování bajtů v paměti a zápisu zadaných hodnot na zvolenou adresu.

4.9. Ověření demonstrační úlohy

Jakmile je ve Vivadu celý návrh vytvořen je možné kliknutím na tlačítko *Validate Design* spustit podrobnou kontrolu návrhu, pokud se objeví informace, že bylo prověření úspěšné *(Validation succesfull)*, může se přejít do procesu implementace a ke kontrole syntaxe, ta vyjde buď správně (objeví se v zelený znak vedle Check syntax) nebo špatně (objeví se červený znak). V SDK proběhne podobná kontrola, a pokud nenajde žádný problém *(error)*, přejde se ke konfiguraci, naprogramování FPGA a následnému spuštění programu kliknutím na *Run*.

Bylo provedeno ověření funkčnosti celého systému, jež byl nakonfigurován na čip SoC Zynq, který se nachází na desce programovatelných hradlových polí ZYBO od výrobce Xilinx, propojené s barevným displejem typu PmodOLEDrgb přes konektor pmod komunikujícím přes SPI rozhraní.



Obrázek 37. Schéma zapojení

4.10. Dosažené výsledky

Základním úkolem této bakalářské práce bylo popsat a navrhnout HW a SW vestaveného zařízení pro řízení barevného organického LED displeje. Výsledkem jsou čtyři funkční simulace aplikací do medicínského prostředí, ve kterých bylo využito co nejvíce různých funkcí k docílení uceleného vzhledu. Na přiloženém *CD* byly pro představu vytvořeny gify vizualizací demonstračních aplikací.

Obrázek 38. Úvodní snímek

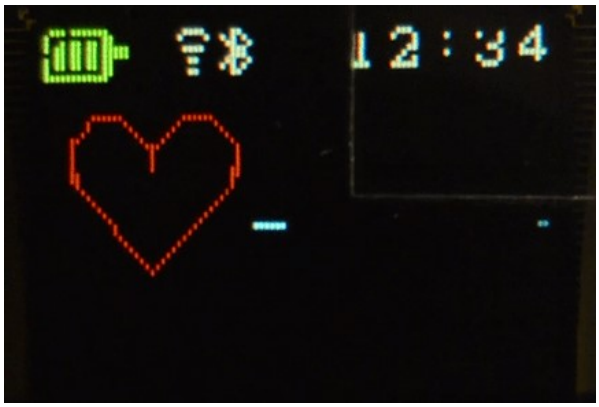
Úvodní snímek

Úvodní snímek představuje statickou vizualizaci, která proběhne jen jednou při spuštění programu.

Obrázek 39. Menu

Menu

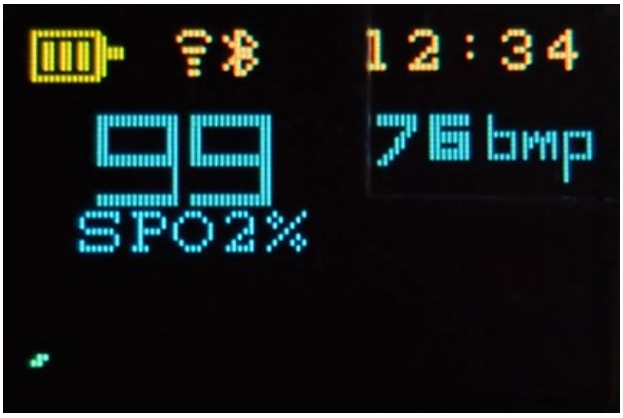
Menu volby se objeví pokaždé, když program nemá zvolenou vizualizaci *(spínač)*, slouží pro výběr animace.



Obrázek 40. Vizualizace EKG

EKG

Vizualizace EKG se spustí vybráním prvního spínače *(SW0),* jedná se o pohyblivou animaci, která postupně vykresluje EKG křivku a tlukot srdce.



Obrázek 41. Vizualizace SpO2

SpO2

Vizualizace SpO2 simulující prostředí prstového pletysmografu se spustí vybráním druhého spínače *(SW1),* jedná se o pohyblivou animaci, která postupně vykresluje tepající srdce a SpO2 křivku. Hodnoty se mění v závislosti na srdci a křivce.

Obrázek 42. Vizualizace Spiro

Spiro

Vizualizace spirometrie je pokus o instruktážní animaci při měření objemu plic. V dolní části se objevují instrukce a v grafu se vykresluje postupně konkrétní objem plic. Animace se spustí vybráním třetího spínače *(SW2),* jedná se o pohyblivou animaci.

Obrázek 43. Vizualizace Lebka

Lebka

Tato animace, jak již název napovídá, zobrazí lebku, která se uživateli ukáže ze všech stran, dává tak pocit 3D zobrazení. Spustí se vybráním čtvrtého spínače *(SW3),* jedná se o pohyblivou animaci.

** Nutno podotknout, že vizualizace jsou reálně méně modré, tento nedostatek byl způsoben nahrávacím zařízením.*

Závěr

Prvním úkolem této bakalářské práce bylo provést zhodnocení a vytvořit tak teoretický základ k technologii organických elektroluminiscenčních součástek (OLED), které, jsou-li v pravidelné matici, tvoří OLED displej, dále pak seznámení s technikou programovatelných logických obvodů, kde bylo provedeno i srovnání s různými typy. V rámci práce jsou dále rozebrána pouze na programovatelná hradlová pole (FPGA). Pro účely vytvoření demonstrační úlohy byl vybrán grafický displej PmodOLEDrgb vyráběný firmou Digilent.

Technologie FPGA byla použita k realizaci demonstrační úlohy, konkrétně v podobě FPGA desky ZYBO s dvou jádrovým čipem z rodiny Zynq 7000, na který bylo vytvořeno konkrétní blokové schéma realizace návrhu hardwaru v programovacím prostředí Vivado. Softwarová část byla napsána v programovacím jazyce C a obsahuje funkce nezbytné pro řízení, komunikaci a hlavně vykreslování na displeji. Program i jednotlivé funkce jsou zde popsány s detailními příklady použití.

Záměrem praktické části bylo vytvořit pohyblivou aplikaci, kterou tvoří čtyři hlavní animace a dva úvodní snímky. Implementace systému byla realizována na několika úrovních. V nejnižší vrstvě byl implementován design hardwaru na FPGA. Na nejvyšší úrovni je cílová uživatelská aplikace, pomocí které je vestavěný systém také ovladatelný tlačítky a spínači. Ke tvorbě této aplikace bylo třeba využít nejen teoretického základu z první části, ale také spoustu času při návrhu jednotlivých jednoduchých obrazců, které byly následně pomocí programu rozpohybovány a zkompletovány do ucelených animací.

Záměrem bakalářské práce bylo nastínit tématiku technologie OLED displejů, na kterých se zobrazují informace zprostředkované uživatelsky konfigurovatelným procesorem na desce programovatelných hradlových polí (FPGA) a také přiblížit postup návrhu hardwaru a softwaru vestavených zařízení. Práce tak poskytuje co nejpřehlednější instruktáž pro vytvoření a pozdější rozvinutí tohoto tématu v kontextu dalších zařízení.

Použitá literatura a zdroje

[1] Encyklopedie fyziky: Typ trinitron. *Fyzika.jreichl* [online]. [cit. 2017-10-10]. Dostupné z: http://fyzika.jreichl.com/main.article/print/1309-typ-trinitron

[2] 8MP Clinical Review Monitor. In: *LG* [online]. LG Electronics, 2009 [cit. 2017-10-10]. Dostupné z: http://www.lg.com/us/business/commercial-display/products/medical-monitors/lg-27HJ712C

[3] CHOI, Mi-Ri, Seong-Hoon WOO, Tae-Hee HAN, et al. Polyaniline-Based Conducting Polymer Compositions with a High Work Function for Hole-Injection Layers in Organic Light-Emitting Diodes: Formation of Ohmic Contacts. *ChemSusChem* [online]. 2011, **4**(3), 363-368 [cit. 2017-10-12]. DOI: 10.1002/cssc.201000338. ISSN 18645631. Dostupné z: http://doi.wiley.com/10.1002/cssc.201000338

[4] Oled-process. In: *Msu.edu* [online]. HowStuffWorks, 2005 [cit. 2017-10-10]. Dostupné z: https://msu.edu/~dunnjam7/oled-process.gif

[5] OLED displeje - využívané principy a varianty. In: *Automatizace.hw*[online]. 2009 [cit. 2017-10-12].]. Dostupné z: https://automatizace.hw.cz/oled-displeje-vyuzivane-principy-a-varianty

[6] Sony product detail page HMS3000MT. In: *Pro.sony* [online]. HowStuffWorks, 2009 [cit. 2017-12-31]. Dostupné z: https://pro.sony.com/bbsc/ssr/product-HMS3000MT/

[7] Sony product detail page PVM2551MD. In: *Pro.sony* [online]. 2009 [cit. 2017-12-31]. Dostupné z: https://pro.sony.com/bbsc/ssr/product-PVM2551MD/

[8] Medical Lighting, will it make a breakthrough for the OLED lighting market? *Olednet* [online]. olednet, 2015 [cit. 2017-12-31]. Dostupné z: http://www.olednet.com/en/medical-lighting-make-breakthrough-oled-lighting-market/

[9] DIGILENT, Inc. [i] *PmodOLEDrgb: 96 x 64 RGB OLED Display with 16-bit color resolution* [/i] [online] Dostupné z: http://store.digilentinc.com/pmod-oledrgb-96-x-64-rgb-oled-display-with-16-bit-color-resolution/.

[10] SSD1331. *Cdn-shop.adafruit* [online]. Solomon Systech Limited, 2007 [cit. 2017-10-12].]. Dostupné z: https://cdn-shop.adafruit.com/datasheets/SSD1331_1.2.pdf

[11] KAPPENMAN, Tommy. *Getting the PmodOLEDrgb to Work on Zybo*. In: [i]Instructables [/i][online] [cit. 30. 6. 2017]. Dostupné z: http://www.instructables.com/id/Getting-the-PmodGPS-to-Work-on-Zybo/

[12] Moderní trendy ve vývoji digitální elektroniky ve strukturách FPGA. *Dps-az* [online]. Ing. Soběslav Valach, DFC Design, 2010 [cit. 2017-12-13]. Dostupné z: https://www.dps-az.cz/vyvoj/id:5501/moderni-trendy-ve-vyvoji-digitalni-elektroniky-ve-strukturach-fpga

[13] KAŠÍK, Vladimír. *Programovatelná hradlová pole FPGA*. Ostrava, 2013. Učební text a návody do cvičení. VŠB – Technická univerzita Ostrava.

[14] Digilent, Inc. *Zybo Zynq-7000 ARM/FPGA SoC Trainer Board (LIMITED TIME)>> see Zybo Z7-10 for replacement* [online]. 2017 [cit. 2017-12-13]. Dostupné z: http://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/

[15] CROCKETT, Louisse H., Ross A. ELLIOT, Martin A. ENDERWITZ a Robert W. STEWART. [i]*The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc.*[/i] Glasgow: Strathclyde Academic Media, 2014. ISBN 978-0992978709. Dostupné také z: http://www.zynqbook.com/.

[16] DVOŘÁK, TOMÁŠ. *OVLÁDÁNÍ VESTAVĚNÉHO SYSTÉMU PŘES INTERNET* [online]. BRNO, 2017 [cit. 2017-12-13]. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/69889/19045.pdf?sequence=2&isAllowed=y. BAKALÁŘSKÁ PRÁCE. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ.

[17] Vývoj aplikací s moduly SQM4: Začínáme s Xilinx Zynq. *Dps-az*[online]. CLIQUO & Binteractive, 2015 [cit. 2017-12-13]. Dostupné z: https://www.dps-az.cz/vyvoj/id:29146/vyvoj-aplikaci-s-moduly-sqm4-zaciname-s-xilinx-zynq

[18] ANDERSON, Paul. [i]Advanced Display Technologies.[/i] JISC Technology & Standards Watch [online] [cit. 30. 6. 2017]. Dostupné z: https://pdfs.semanticscholar.org/1dda/2d638a42a6ecea239ec2dc72f196f4111fb4.pdf.

[19] XILINX, Inc. [i] *Vivado Design Suite Tutorial* [/i][online]. [cit. 30. 6. 2017]. Dostupné z: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug995-vivado-ip-subsystems-tutorial.pdf

[20] Barevné spektrum. In: *Chytre-bydleni* [online]. 2017 [cit. 2018-01-25]. Dostupné z: http://www.chytre-bydleni.cz/w/chytrebydleni

[21] ŠŤASTNÝ, J. FPGA prakticky: Realizace číslicových systémů pro programovatelná hradlová pole. 1. vydání. Praha: BEN – technická literatura. 2010. 200 s. ISBN 978-80-7300-261-9.

[22] DOLEČEK, J. Moderní učebnice elektroniky, 3. díl. Optoelektronické prvky a optické vlákna. Ben - Technická literatura, Praha 2005. ISBN 80-7300-184-5.

[23] SPI rozhraní. In: *Bulletin* [online]. 2014 [cit. 2018-04-22]. Dostupné z: https://forum.pjrc.com/threads/24993-SPI-Data-in-out-naming-clarity

[24] SPI rozhraní. In: *Tajned* [online]. 2016 [cit. 2018-04-22]. Dostupné z: http://www.tajned.cz/2016/12/spi-rozhrani/

[25] *Product Specification: OLED Display Module* [online]. Taiwan: Univision Technology, 2006 [cit. 2018-04-25]. Dostupné z: https://cdn-shop.adafruit.com/datasheets/UG-9664HDDAG01.pdf

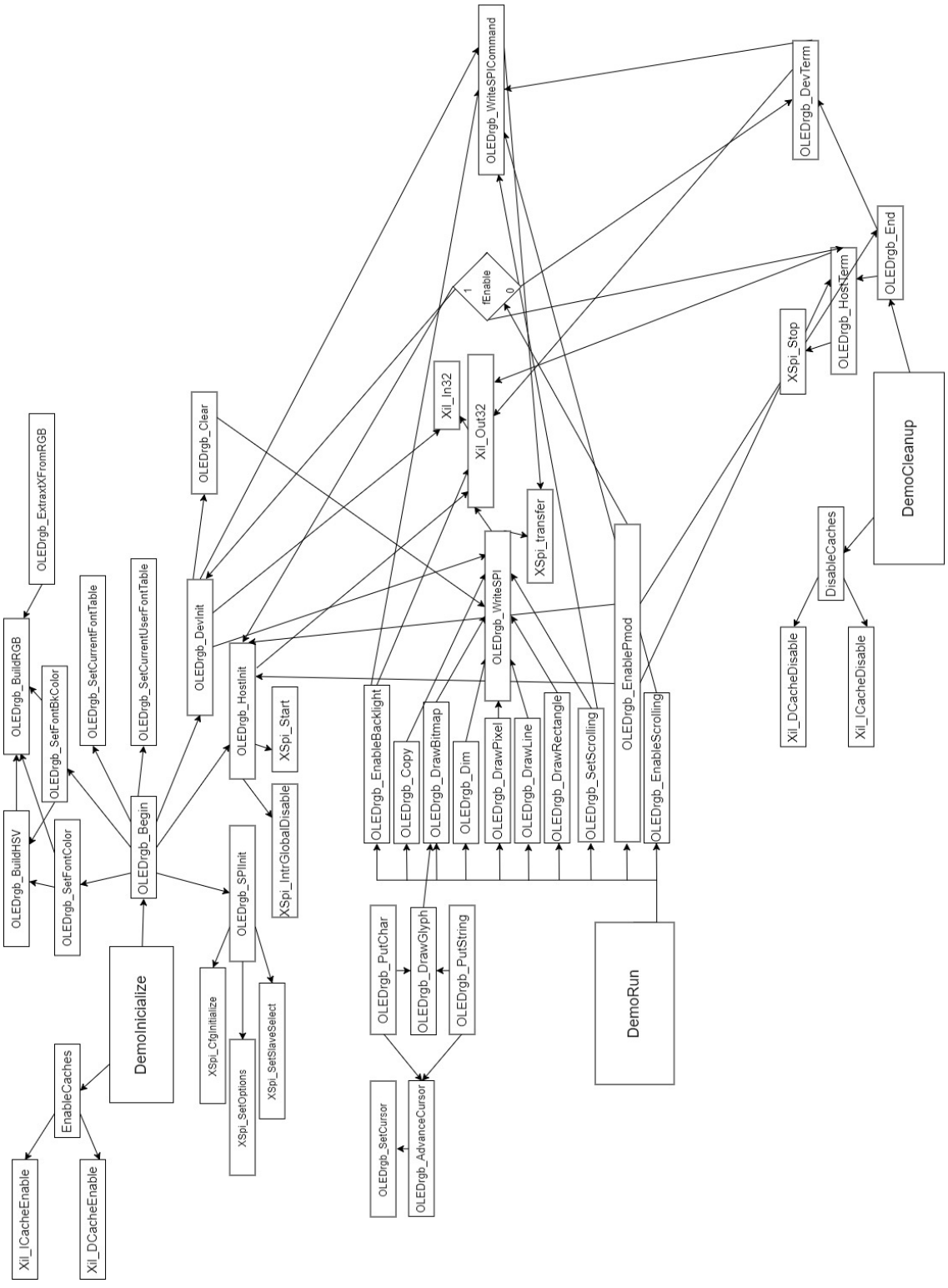
Seznam příloh:

- 1. Příloha: Mapa návaznosti funkcí
- 2. Příloha: Návod k vytvoření programu

Obsah přiloženého CD:

- Bakalářská práce
- Vytvořený program
- Gify jednotlivých animací
- Návod k vytvoření programu
- Mapa návaznosti funkcí

1. Příloha:



Obrázek 44. Mapa návaznosti funkcí

2. Příloha:

Návod k vytvoření projektu na ZYNQ 7000 a PmodOLEDrgb displeji

Stáhnout a instalovat soubor definující komponenty desky a IP repositář

Digilent board files – Obsahuje definice deskových pinů pro několik typů desek Digilent (včetně ZYBO)
Knihovna Digilent IP – Obsahuje několik IP adres navržených pro práci s Digilent hardwarem.

Obě tyto knihovny jsou uloženy v Digilent Githubu a jsou pravidelně aktualizovány. Je nutné mít nejnovější verzi aplikace Vivado s nainstalovanou sadou Xilinx SDK.
Nyní je nutné extrahovat vivado-boards a zkopírovat obsah new / board_files do
C: /Xilinx/Vivado/2015.4/data/boards/board_files

Případně, vytvořit tcl skript ve složce Xilinx appdata. Je možné najít zadáním %appdata% v okně vyhledávání. Ve složce Roaming je třeba přejít do Xilinx / Vivado a vytvořit nový textový soubor v této složce s názvem init.tcl, pak zkopírovat a vložit následující řádek a uložte jej set_param board.repoPaths [seznam "C: / [path to vivado-boards / new]"]. Tím se nainstalují soubory desek pro všechny verze Vivado.

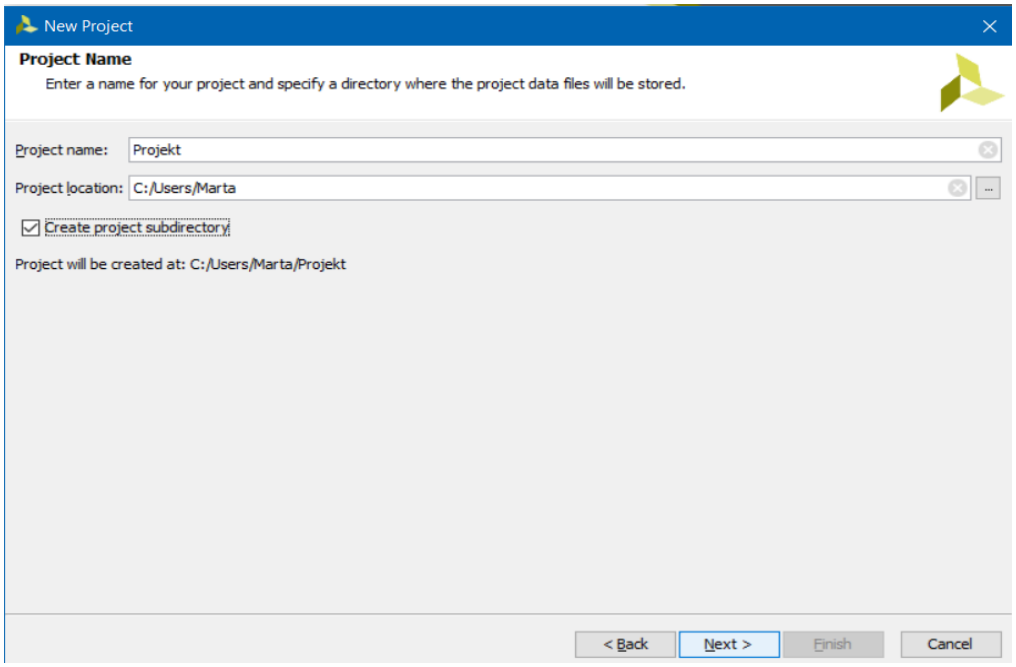
Krok 1: Vytvoření nového projektu

Otevřít Vivado:

- Vytvořit nový projekt kliknutím na **Create New Project**

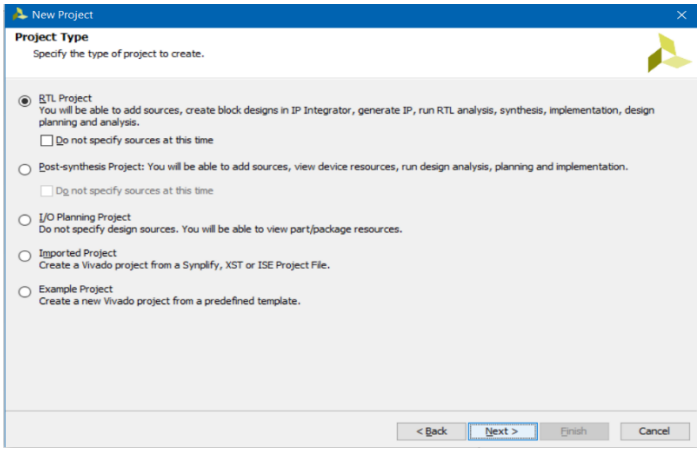
Obrázek 45. Vytvoření nového projektu

- Otevře se další stránka Project Name
  - Pojmenovat projekt
  - Určit uložiště
  - Ujistit se, že **Create project subdirectory** je označen a kliknout na **Next**



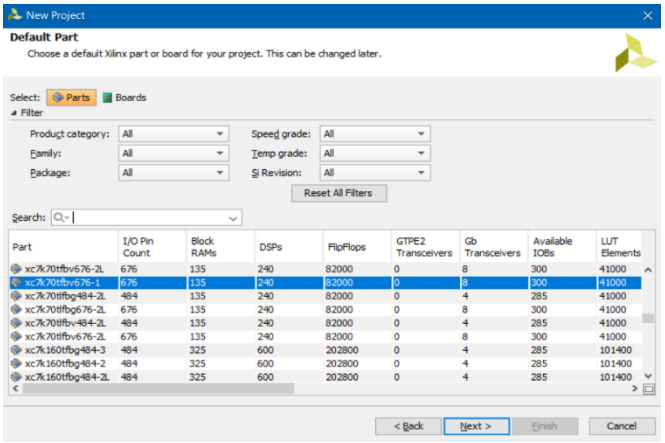
Obrázek 46. Pojmenování nového projektu

- Na stránce Project Type, vybrat **RTL Project** a kliknout na **Next**

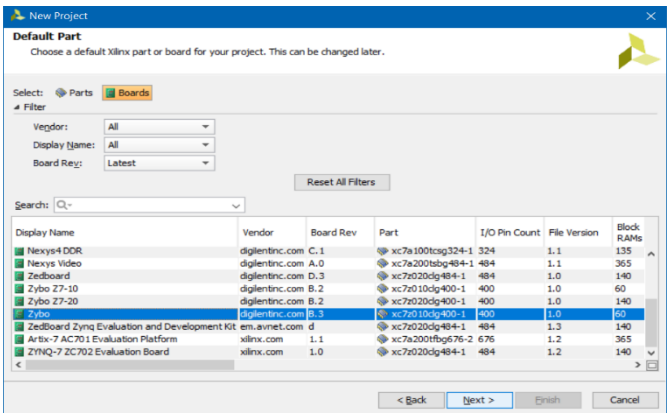


Obrázek 47. Typ projektu

- V okně **Add Sources** zajistit, že **Targert language** je nastaven na **VHDL** kliknout na **Next**
  - Zdroje se přidají později, využitím konstrukčního plátna ve Vivado
- V **Add Existing IP** klik na **Next** a v **Add Constraints** opět **Next**
- V dialogovém okně **Default Part** označit možnosti a kliknout na **Next**
  - Vybrat v **Parts → xc7k70tfbv676-1**
  - Vybrat v **Board → ZYBO**

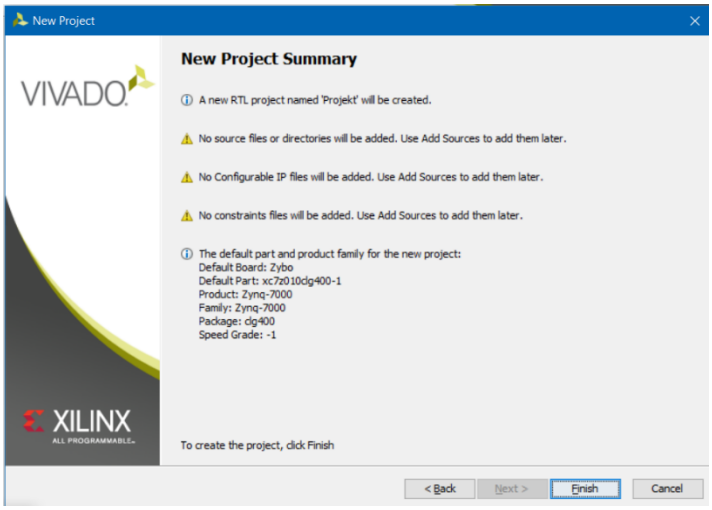


Obrázek 48. Výběr výchozích částí



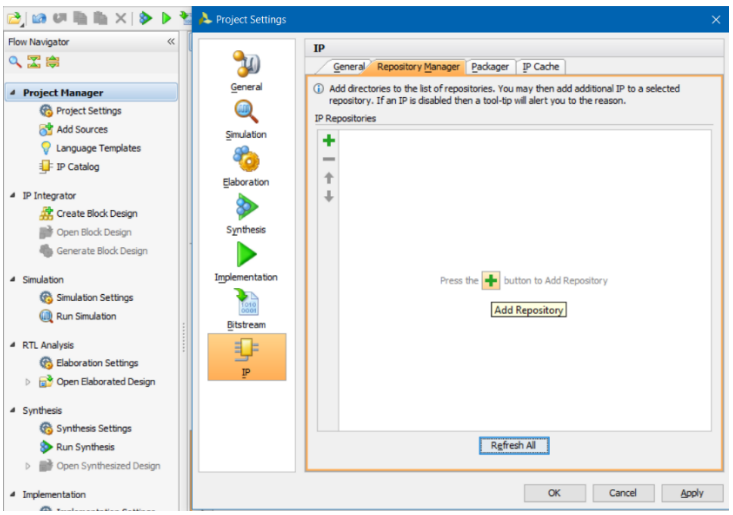
Obrázek 49. Výběr desky

- Přehled shrnutí nastavení nově vytvořeného projektu je v **New Project Summary**
  - Kliknout na **Finish** pro vytvoření nového projektu



Obrázek 50. Shrnutí nového projektu

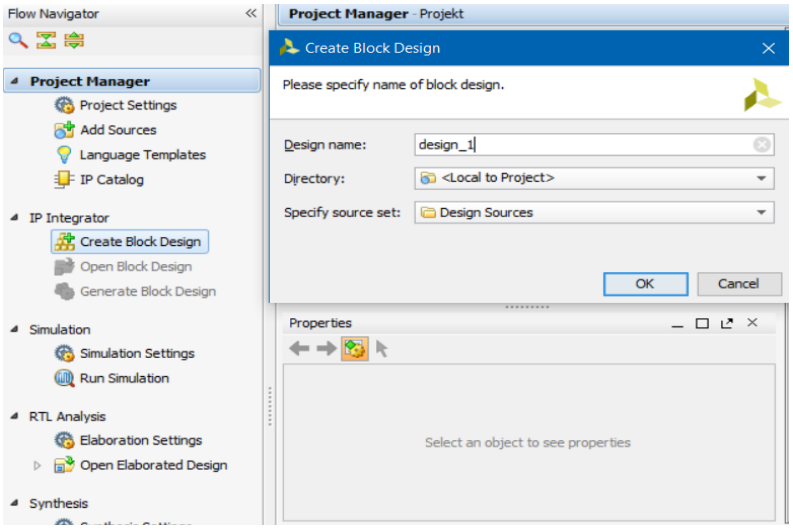
Krok 2. Přidání Digilent IP repositáře



Obrázek 51. Přidání IP repositáře

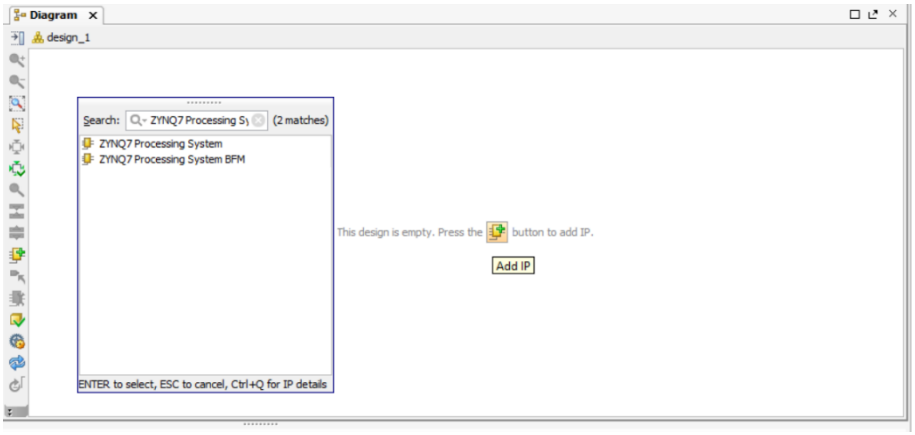
Uvnitř nového projektu v aplikaci Vivado klikněte na položku **Project Settings**  , která se nachází vlevo pod **Project Manager**. Klepněte na **IP** a vyberte kartu **Respiratory Manager**. Klikněte na + a přejděte do složky extrahované knihovny Vivado a vyberte ji před stisknutím tlačítka **OK**.

Krok 3: Vytvoření nového Block Design



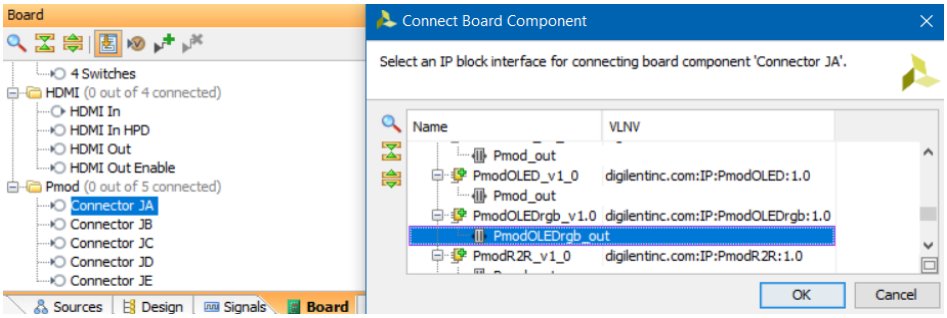
Obrázek 52. Vytvoření nového Block Design

- Vlevo v části **IP Integrator** klik na tlačítko **Create Block Design** (název musí být bez mezer)



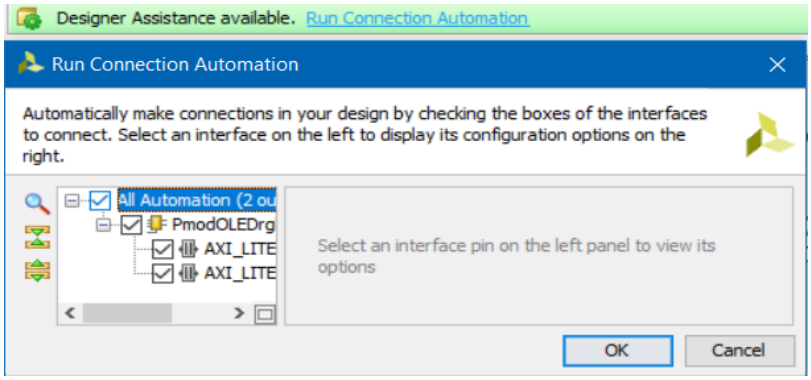
Obrázek 53. Přidání procesoru

- Poté kliknout na tlačítko **Add IP**, které se nachází uprostřed nového designu.
  - Zde vybrat **ZYNQ7** a přidat ho do návrhu.
- Vlevo od návrhu je karta **Board**
  - Otevřít konektor **Pmod**, ke kterému bude připojen PmodOLEDrgb. Zde je nutné vybrat adresu **PmodOLEDrgb_v1.0** a kliknout na tlačítko **OK**.



Obrázek 54. Vybrání Pmod

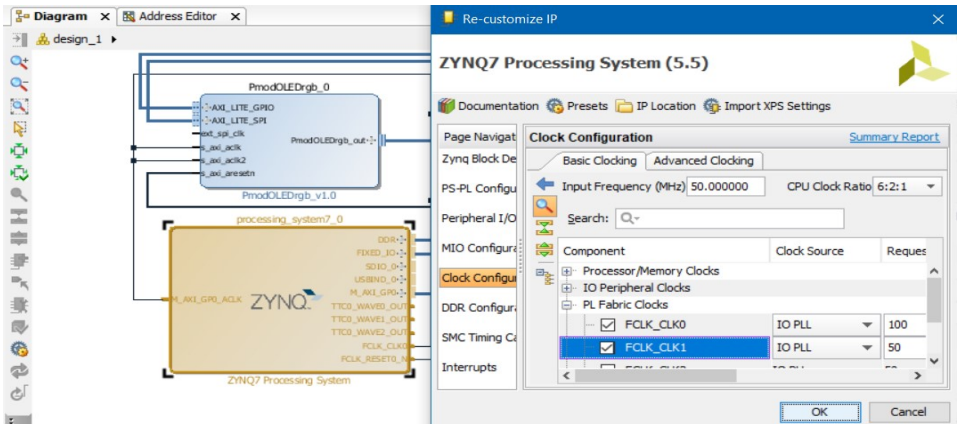
- Kliknutím na tlačítko **Run Block Automation** se otevře okno, které se potvrdí tlačítkem **OK**. Potom je třeba kliknout na tlačítko **Run Connection Automation** a vybrat možnost **All Automation** a potvrdit tlačítkem **OK**.



Obrázek 55. Automatické navázání cest

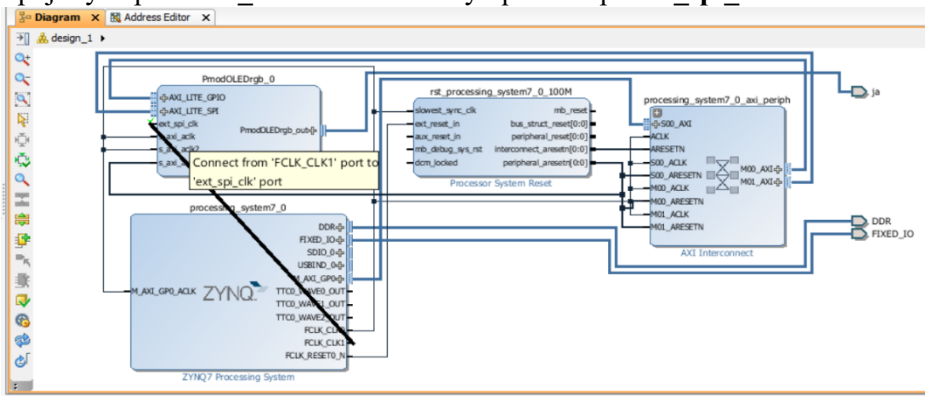
Krok 4: Přidání hodin

- Poklepáním na blok **ZYNQ** se otevře jeho nastavení.
  - Kliknutím na položku **Clock Configuration**, se pak rozbalí nabídka **PL Fabric Clocks** a zde je nutné vybrat ještě druhé pole a potvrdit tlačítkem **OK**.



Obrázek 56. Výstup FCLK_CLK1

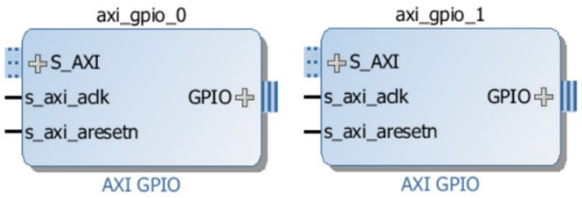
- Poté se spojí výstup **FCLK_CLK1** na bloku Zynq ke vstupu **ext_spi_clk** na bloku PmodOLEDrgb



Obrázek 57. Propojení výstupu FCLK_CLK1 a ext_spi_clk

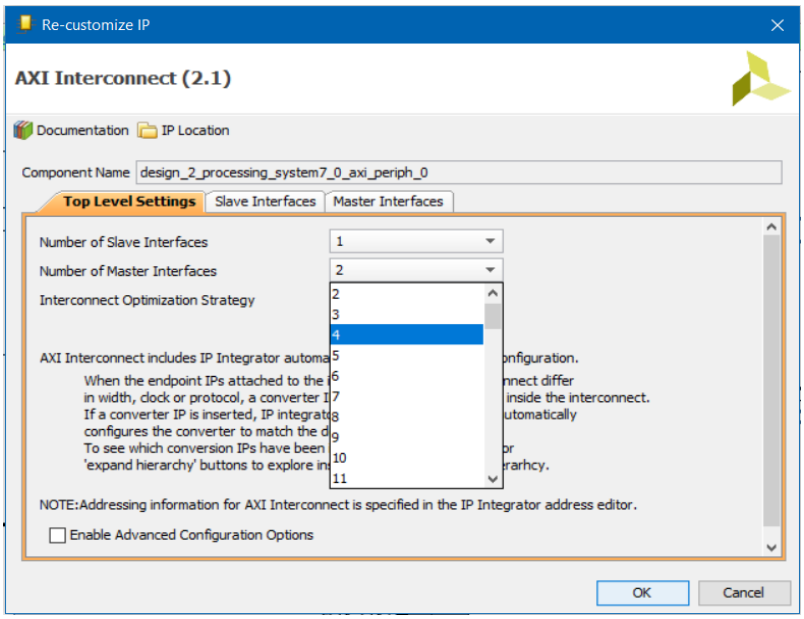
Krok 5: Přidání, propojení a nastavení bloků GPIO_0, GPIO_1

- Tlačítkem  přidáme dva další bloky – AXI GPIO



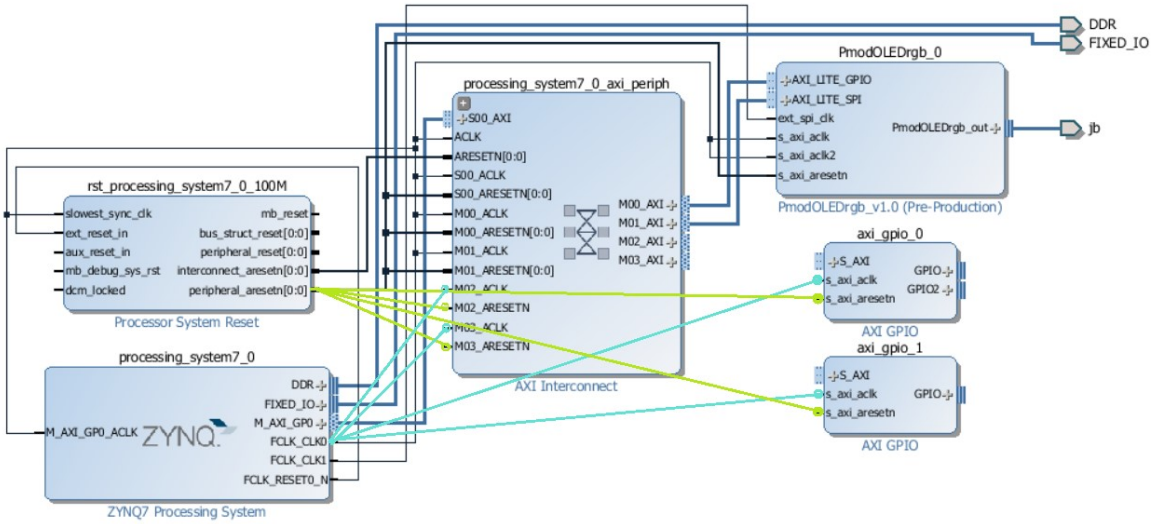
Obrázek 58. Přidání GPIO bloků

- Poklepáním na blok **AXI Interconnect** se otevře jeho nastavení.
  - V záložce **Top Level Settings** je nutné zvýšit počet Master rozhraní na 4.



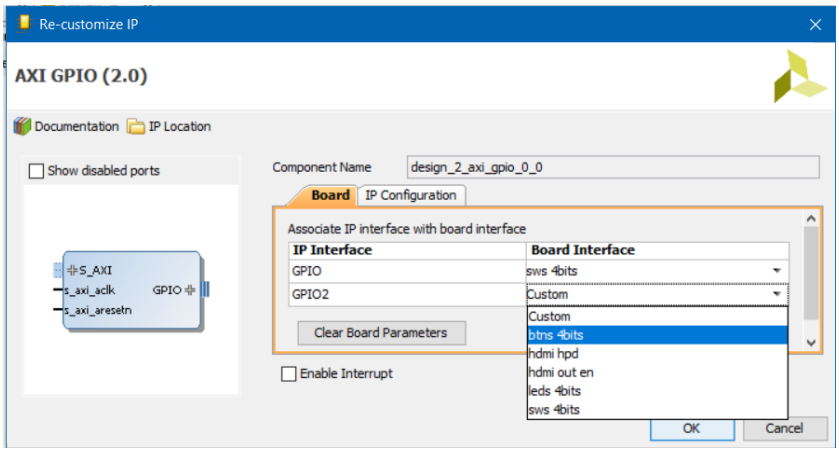
Obrázek 59. Re-customizace IP – AXI Interconnect

- Nyní je třeba bloky správně propojit.
  - **Peripheral_aresetn** se propojí s **s_axi_aresetn** u obou GPIO bloků a s **M02_ARESETN** a **M03_ARESETN** na AXI rozhraní
  - **FCLK_CLK0** se propojí s **M02_ACLK**, **M03_ACLK** a také s **s_axi_aclk** opět u obou GPIO

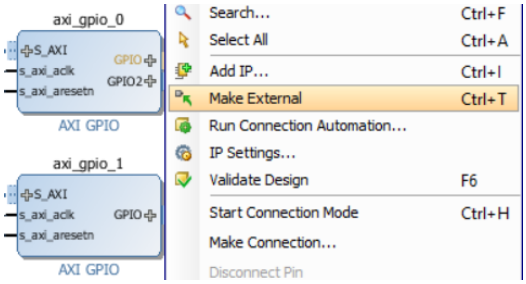


Obrázek 60. Propojení bloků GPIO

- Dále je nutné propojit bloky GPIO s piny na desce v záložce **Board** v okně, které se otevře po kliknutí na AXI GPIO blok a poté **vytvořit výstup** kliknutím na GPIO pravým tlačítkem na myši.

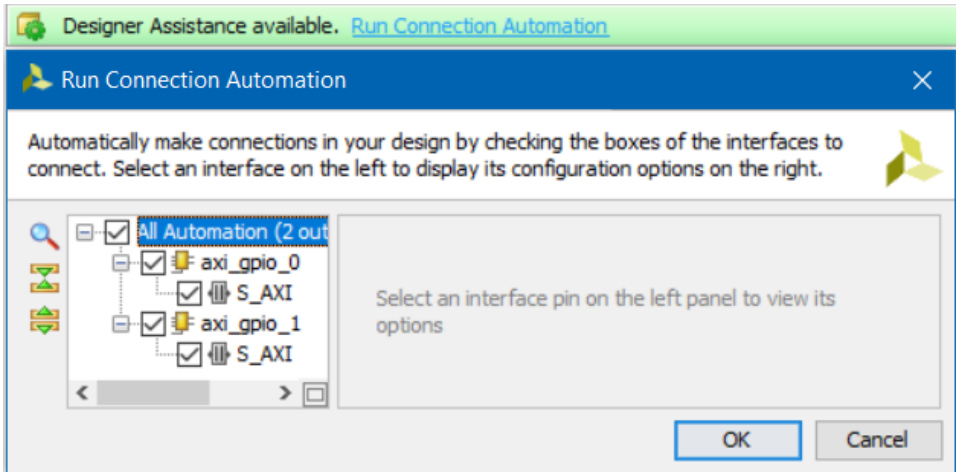


Obrázek 61. Re-customizace – AXI GPIO



Obrázek 62. Vytvoření výstupu

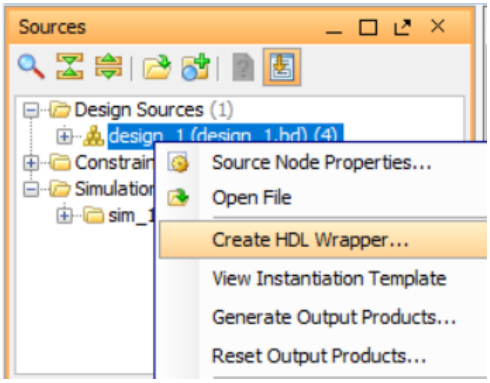
- Kliknutím na tlačítko **Run Connection Automation** a vybráním možnosti **All Automation** a potvrzením tlačítkem **OK** se bloky propojí. Tlačítkem se přehledně bloky uspořádají a se uzpůsobí velikosti Block Design okna. Případně je možnost přiblížit části , či je naopak oddálit



Obrázek 63. Automatické propojení

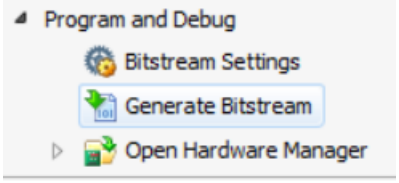
Krok 6: Vytvoření obálky a vygenerování bitstreamu

- **Sources** vlevo od návrhu bloku
  - Kliknutím pravým tlačítkem myši na návrh se rozvine menu, zde kliknout na **Create HDL Wrapper** a nechat Vivado zvládnout vytvoření obalu HDL



Obrázek 64. Vytvoření obálky

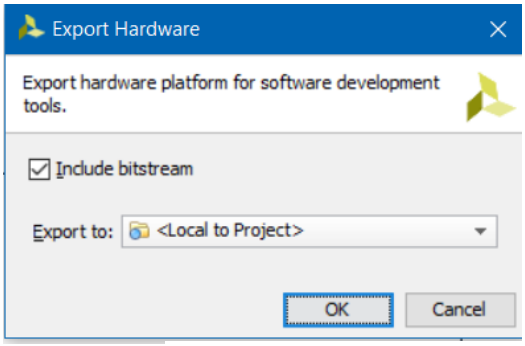
  - Poté kliknutím na tlačítko **Generate Bitstream**, které se nachází pod programem a laděním, poté se objeví upozornění, že nejsou k dispozici žádné výsledky implementace, zde je vhodné vybrat **Yes**



Obrázek 65. Vygenerování Bitstreamu

Krok 7: Přesunutí do Xilinx SDK

- V menu **File>Export>Export Hardware** a kliknout na tlačítko **OK** a vybrat pole **Include bitstream**
- Poté v příkazu **Soubor> Spustit SDK** a potvrdit tlačítkem **OK**, pak se otevře projekt v Xilinx SDK

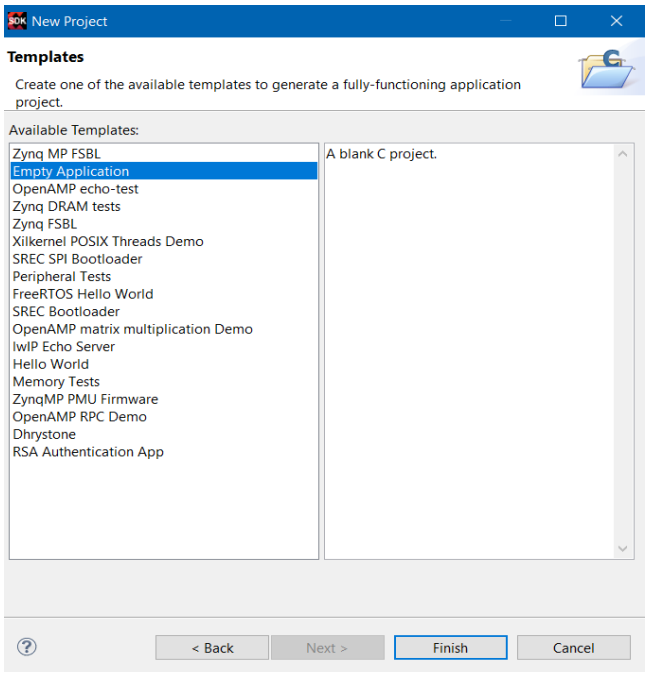


Obrázek 66. Export

Krok 8: Vytvoření nové aplikace a import do zařízení

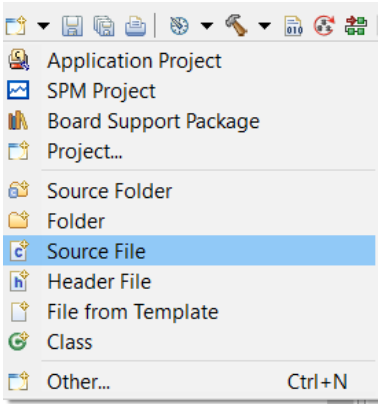
V sadě Xilinx SDK:

- Malou černou šipkou vedle položky New se otevře menu a zde vybrat **Application Project**.
- Musí být nějak pojmenován a poté tlačítko **Next**
- V seznamu dostupných šablon vybrat možnost **Empty Application** a poté na tlačítko **Finish**



Obrázek 67. Seznam dostupných šablon

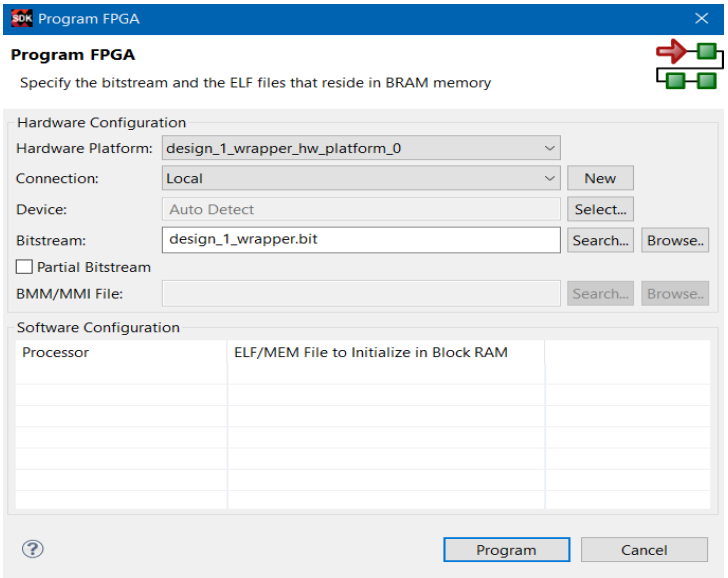
- V menu černé šipky vybrat možnost **Source File** a pojmenovat main.c, pak tento soubor uložit do podložky **src** nově vytvořeného projektu
- Zde se přidají knihovny a funkce tedy celý vlastní program



Obrázek 68. Vytvoření hlavičkového souboru

Krok 9: Naprogramování FPGA

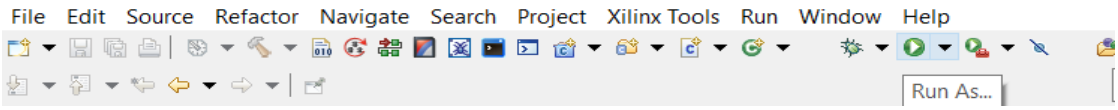
- S napsaným programem
  - Na horním panelu nástrojů vybrat **Xilinx Tools > Program FPGA**. Zde se zkompilují bitstream vytvořený ve Vivadu a konfigurace software vytvořená v SDK, kliknutím na **Program FPGA** se tímto zkompilovaným souborem naprogramuje FPGA



Obrázek 69. Programování FPGA

Krok 10: Spuštění programu

V tuto chvíli na FPGA desce svítí LED značící nakonfigurování, takže je nezbytné program spustit. Na listě v záložce **Run as** je potřeba vybrat **Launch on Hardware (System Debugger),** čímž se program spustí.



Obrázek 70. Spuštění programu